POLITECNICO DI TORINO

Master's degree in
Data Science and Engineering

Thesis

Generative AI for the Simulation of Autonomously Driven Vehicles

Supervisors
Claudio Casetti
Diego Gasco
Supervisors' signature

. . . . . .
. . . . . .

Candidate
Sergio Augusto Angelini

Candidate's signature

. . . . . .

Academic Year 2023-2024

Summary

In recent years, growing amounts of research have been devoted to the field of autonomous vehicles, reaching exceptional results and foreshadowing the advent of an era when cars will be capable of navigating the roads completely on their own. The body of research presented in this thesis falls within the scope of this technological revolution, as it explores the possibility of generating custom risk scenarios for the simulation based training of autonomous driving systems. To this end we first introduce a custom, effective event classification system based on gravity measures developed by the traffic research community. We apply this scheme to a benchmark road network developed in the Simulation of Urban Mobility SUMO, obtaining a dataset of configurations that lead, in probability, to collision events. The dataset is then used to train a Wasserstein conditional generative adversarial network that generates the required risk scenarios. We discuss in detail the strategies needed to obtain an effective generator and show how the imbalances in the classes' distribution within the dataset can be mitigated. Furthermore, we demonstrate that it is indeed possible to obtain a converging generator and perform a statistical assessment of its performance to prove its effectiveness.

Contents

List of Tables

List of Figures

Mathematics is a game played according to certain simple rules with meaningless marks on paper.

[D. Hilbert]

Chapter 1

Introduction

Over the past decade, significant progress has been made towards the realization of autonomous driving vehicles. Automotive manufacturers have progressed to remarkable achievements, providing systems capable of advanced autonomous features such as independent navigation within urban environments. Recently, SAE level 3 conditional automated driving systems have been certified, allowing the automated driving function control over multiple driving tasks. However, a fallback-ready user is still required and a number of limitations hinder the autonomous capabilities characterizing SAE levels 4 and 5. As the required complexity of such systems grows, the issues raised by development and testing correspondingly increase. Manufacturers differ in the strategies they employ to address these problems; although live data collection from either selected vehicles or whole fleets remains the fundamental strategy employed to train autonomous systems. This method, albeit successful, implies a dependence on live data and it is unclear whether systems trained on a limited number of scenarios possess the generalization capabilities required to perform adequately on unseen situations. Considerations of this kind form a compelling argument in favour of the adoption of simulated environments for the training and testing of autonomous driving systems. Indeed, simulation based training would enable developers to have full control in defining repeatable, completely customized development and test scenarios. Furthermore, simulation based training would lessen the implicit bias present in systems trained on limited regions, as it grants access to world scale infrastructure data. In this direction, open source simulators such as Simulation of Urban Mobility SUMO and CARLA [22, 7] represent ideal candidates for an integrated system capable of satisfying the outlined requirements. Overall, both frameworks allow researchers to simulate traffic conditions in realistic urban environments. Sumo's development began in the early 2000s, when the need for an open source microscopic road traffic simulator arose in the context of the traffic research community. During the past 20 years, SUMO has been successfully

employed to study the large scale dynamics of traffic systems, becoming an important tool for road networks optimization [21, 24] and the natural candidate for the objectives previously outlined.

The purpose of this thesis work is to provide a scalable framework for the generation of criticality scenarios within SUMO for the training of autonomous vehicles. Therefore, we have divided the research into three main areas. We first propose a comprehensive scenario classification system and then define a benchmark road network and use the developed system to categorize the observed SUMO outputs. This task is repeated to build a dataset of initial conditions that, once implemented in SUMO, should lead to the observed event. The third and final part of the research focuses on the development of a Generative Adversarial Network capable of generating initial conditions based on user-specified gravity levels.

Crucially, the seminal challenge in the development of this work has been the definition of a sensible classification approach. Indeed, an acceptable scheme must posses a number of properties such as, for example, independence to the network's geometry, sufficient granularity and differentiation between agents. The answer to these issues is discussed in Chapter 3, where we present an approach based on current research concerning fatality rates in traffic collisions.
We then apply the proposed scheme to a benchmark road network topology implemented in SUMO. In Chapter 2 we introduce the Simulation of Urban Mobility and present an in depth discussion of the selected parameters and specify the architecture employed in the simulations. Consisting of a four way intersection and independent pedestrian crossings, our proposal is designed to be computationally efficient while retaining the necessary level of complexity to generate realistic scenarios.
Chapter 4 recounts the implementation of the generative model, explaining in detail the theory underlying Generative Adversarial Networks and their training. Furthermore, we discuss the definition of suitable losses and regularization schemes that have proved crucial to reach effective models. Generating synthetic initial conditions presents unique challenges, as the data must adhere to strict requirements to be considered realistic. Present work shows how imposed constraints can be learned by the generating agent by means of strongly regularizing losses, and a general approach is proposed to allow the application of the framework to road networks of arbitrary complexity.
Chapter 5 conveys the experimental results that validate the proposed approach, providing a statistical analysis of the effectiveness of our methods and discussing the impact of parameters specification on the observed results. The findings from this chapter provide valuable insights into the strengths and limitations of the

framework, contributing to the broader body of research on critical scenario generation for autonomous vehicle training. Finally, Chapter 6 outlines future research directions, discussing possible generalizations and extensions of this research.

Chapter 2

Simulation of Urban Mobility

Simulation of Urban Mobility, hereby known more succinctly as SUMO, is a microscopic road traffic simulation. Its development began in 2001 to support the traffic simulation community with an open-source tool where algorithms could be implemented [4]. At the time many other simulations were available, but the almost non-existent comparability of the implemented models and algorithms caused major drawbacks. Issues in portability, reproducibility, and general maintenance of the simulators proved the necessity of a unified framework. The major part of SUMO's development is undertaken by the Institute of Transportation Systems at the German Aerospace Center [25] (Deutsches Zentrum für Luft- und Raumfahrt, DLR), with several external partners supporting different extensions to the simulation suite. In the following sections, we will begin by illustrating the fundamental components of the simulation. The internal dynamics of the system will be discussed in Section 2.1. Lastly, we will focus on the road network designed and developed for the simulations.

2.1 The suite

Currently, the integrated SUMO suite consists of more than 150 tools. They cover topics ranging from traffic network analysis, demand generation, and demand modification to output analysis. This brief section provides an overview of the main components of the suite that have been employed in the development of the present work.

2.1.1 Sumo

Sumo is the simulation itself; it is a microscopic, space-continuous, and time-discrete traffic flow simulation [22, 4]. It is the core component of the suite, written in C++ and command-line operated, although a graphical user interface is available, called "sumo-gui". To simulate a well-defined scenario, SUMO requires as minimal inputs a road network file and a set of routes. In this limited scenario, no agent (vehicle, pedestrian, or cyclist for example) is present and the simulation describes no evolution for the duration of the selected time frame. The road network file conveys the part of a map related to traffic, the roads, and the intersection the simulated vehicles run along or across. On an abstract level, it represents the directed graph underlying the map. In SUMO's language, intersections are known as junctions and roads or streets as edges. Aggregating these two core components leads to a complete representation consisting of streets (edges) as collections of lanes, traffic light logic referenced by junctions (including right-of-way regulations) and connections between lanes at junctions. Consequentially, the allowed paths for each agent within the simulation are described in the route file. The editing of a road network can be completed using either the "netconvert" or the "netgen" tools. Netconvert is a command line conversion tool compatible with a plethora of allowed network formats, including OpenStreetMap, VISUM, and OpenDRIVE. It allows the user to import existing road networks from real-world scenarios by converting them into XML files compatible with the simulation. Netgen, on the other hand, allows the user to define completely custom networks via a drag-and-drop graphical user interface. In the present work we employed netgen to develop a custom geometry to serve as a benchmark for the proposed framework. In order to populate the road network with agents, sumo requires the specification of traffic demand. Specifically, trips are defined as vehicle movements from one place to another denominated by the starting and arriving edges and the departure time. The collection of edges traversed by the agent is called route, and is the fundamental characterization required by sumo to generate agent's movements. Currently, the sumo-package contains four applications for generating routes. Duarouter is the tool responsible for importing routes or their definitions from other simulation packages and to compute shortest paths using Dijkstra's algorithm. Furthermore, in combination with the simulation it can compute the dynamic user assignment formulated by C. Gawron. Moreover, if statistical data is available, the usage of "jtrrouter" may be used to specify flows and turning preferences at junctions. Indeed, jtrrouter is widely employed in the context of large scale demand modeling studies, where networks having the scale of whole cities may be considered. Whenever origins and destinations are the only available data, od2trips is used to convert origin-destination matrices into trips. The fourth tool, "dfrouter", is employed to compute routes from given observation point measures. In this thesis

work the demand generation for the custom benchmark road network has been specified using duarouter. Details will be addressed in the following sections.

2.2 Agent models

Once the actors have been specified, their dynamics are computed by the simulation following prespecified agent models. In this direction, the simplest behavioral models provided by sumo are the ones that apply to pedestrians. Indeed, the framework allows the specification of three different pedestrian models: "noninteracting", "striping", and "jupedsim". In the noninteracting mode, pedestrians walk bidirectionally along normal edges and "jump" across intersections, so that no interaction between pedestrians and vehicles or other pedestrians takes place. They may either be configured to complete a walk in a fixed amount of time or to move along the edges at a fixed speed. Although essential and unrealistic, this mode boasts high execution speeds and can therefore be of use if the individual pedestrian dynamics are not important. The default pedestrian's agent model in sumo is striping. In this scheme, the model assigns 2D coordinates within a lane (sidewalk, walkingarea, or crossing) to each pedestrian. These coordinates are defined relative to the leftmost side of the start of the lane and are updated at every step of the simulation. We will see that this coordinate system differs from the one employed for vehicles, which has, in general, only 1D coordinates within their respective lane. However, pedestrians advance within the lane along the shortest path towards the next junction in the network. The fact that walking areas are not unidirectional implies that the natural direction of the lane may not be the walking direction of the pedestrian. In this mode, pedestrian take the shortest path to the end of the lane, implying that their x coordinate either monotonically increases or decreases. As soon as the end of the lane has been reached, the pedestrian is placed on the next lane. The crucial difference between the striping and noninteracting modes is the presence of collision avoidance mechanisms. To achieve this result, lanes where pedestrians are allowed are subdivided into sections of constant width. The width of the sections is a user-configurable parameter having a default value of $0.65m$. Using this strategy, the problem of collision avoidance reduces to maintaining sufficient distance within the same stripe. Whenever a pedestrian comes too close to another pedestrian within the same stripe it moves in the y-direction (laterally) as well as in the x-direction to change to a different stripe. The y-coordinate changes continuously which leads to situations in which a pedestrian temporarily occupies two stripes and thus needs to ensure sufficient distances in both. The selection of the preferred stripe relies on a direction of motion algorithm preferring evasion to the right for oncoming pedestrians and the expected distance the pedestrian will be able to walk in the stripe without a collision. During a generic simulation

step, each pedestrian moves as fast as possible while still avoiding collisions. The updates are computed in a single pass for each walking direction with the leading pedestrian being updated first and the other following based on their position in the queue. As a consequence of the above movement rules, pedestrians tend to walk side by side on sidewalks of sufficient width. They wait in front of crossings in a wide queue and they form a jam if the inflow into a lane is larger than its outflow. This is the agent model that this thesis work employs to describe pedestrians. Finally, the model jupedsim [19] concludes this brief overlook of the main pedestrian agent models supported by SUMO. Developed by the Jülich Research Center, it is based on advanced social force models and has been coupled to SUMO. It differs fundamentally from the two default modes provided by the simulator as it implements various decision-making processes in the agents' route choice such as the Generalized Centrifugal Force Mode or the Social Force Mode, aimed at replicating real-world dynamics. Although certainly more complex and, possibly, representative we have refrained from using it in our benchmark. This choice is sensible, to provide realistic dynamics the jupedsim model requires the specification of parameters that should be derived from a real-world counterpart of the selected network.

2.2.1 Car Following Models

On the topic of vehicles, sumo implements a variety of car-following and lane changing models. Concerning car following, SUMO bases its description on a seminal paper by Wagner et al. [29], where evidence is given to support the theory that a linear model based on three parameters is capable of describing car-following behavior with great accuracy. Moreover, SUMO supports a variety of car-following models, all sharing three fundamental parameters: τ, actionStepLenght, and Reaction time. Reaction time refers to the time a driver takes to process information and react accordingly. Usually, this value corresponds to the simulation's step length; the decision-making frequency can be modified by setting the actionStepLength parameter. Indeed, actionStepLength is used to decouple the simulation's step length from the frequency of driver decision making and thereby delay or quicken reactions based on the selected step length. The parameter τ is intended to model a driver's desired minimum time headway in seconds. In practice, drivers attempt to maintain a minimum time gap between the rear bumper of the vehicle that precedes them and their own. Larger instances of τ lead to safer driving when used in conjunction with short actionStepLengths, for drivers are able to immediately react to variations in their relative positions within the lane. Conversely, when τ is set to a value smaller than the reaction time unsafe driving is promoted, as drivers are prone to high decelerations or even collisions. By default, car following models will adapt their driving speed to limit necessary braking to a maximum configured

by the decel attribute. Various models deal differently with the threshold defined by the attribute: the default Krauss model, for example, sets it as a hard limit, whereas for other models such as "IDM" the bound is less strict. In the presented work we employed Krauss's model to describe the car following behavior of the simulation's vehicles. Developed by Krauss in 1997 [20], the microscopic, space-continuous car-following model that takes his name is based on the notion of safe speed. Safe speed refers to motor vehicle drivers choosing speed at the speed limit in law to ensure safe driving according to circumstances, road conditions, and traffic conditions. To ensure driving safety on the lane, the velocity of vehicles ought to be controlled within the bounds specified by safe speed. In the original paper, the author considers uniform acceleration and deceleration amongst different vehicles, however, SUMO implementation developed by Thomas Mayer and Daniel Krajzewicz differentiates between individual agents. Indeed, the default SUMO model allows for user-specified acceleration and deceleration. This fact is of crucial importance when considering the aim of the presented work: to provide an effective framework for the generation of event-correlated initial conditions. From a mathematical viewpoint, the safe speed in Krauss's model takes the form:

$$v_{safe} = v_l(t) + \frac{g(t) - v - l(t)t_r}{\frac{v_l(t)+v_f(t)}{2b} + t_r}. \tag{2.1}$$

Where $v_l(t)$ is the speed of the leading vehicle at time t, $g(t)$ is the gap to the leading vehicle in time t, t_r is the driver's reaction time and b is the maximum deceleration of the vehicle. Notably, v_{safe} may be larger than the maximum speed allowed on the road or even larger than the vehicle is capable to reach due to its acceleration. In this case, the actual safe speed is computed as:

$$v_{actual} = \min\{v_{max}, v + adt, v_{safe}\}.$$

Where in this instance dt is the step duration of the simulation.

2.2.2 Lane-Changing Model

In the brief overview concerning car following models, we have purposely avoided discussing the dynamics underlying lane-changing in SUMO. Indeed, lane-changing is a topic of notable complexity and would be material for a thesis on its own [9]. Furthermore, the road network developed and in the context of our simulations does not implement such mechanics (multi-lane roads are not considered), so that this subsection is included purely for the sake of completeness. The lane-changing model in SUMO has been under continuous development since the start of the project in 2001 [citare SPRINGER-SUMO] and it is one of the most crucial aspects of the simulation. Differently from other microscopic lane-changing models, sumo's one explicitly differentiates among four different motivations for lane-changing:

- strategic change,
- cooperative change,
- tactical change and
- regulatory change.

At runtime, a strategic change occurs whenever a vehicle has to change its lane in order to be able to reach the next edge in its route. In practice, this happens every time the current lane of the vehicle is not directly connected with the next edge of the route. The standard sumo terminology refers to the current lane as *dead lane*. Notably, this is somewhat of a misnomer, as the lane does not have to be a dead end to be considered a dead lane: for example, a right-only turn lane is dead from the perspective of a vehicle that needs to go left or straight. Nevertheless, to compute an effective trajectory in the network, vehicles need to choose a sequence of lanes to follow. SUMO's model allows for a reasonable amount of flexibility in this context as generally, for extensive networks, a variety of possible trajectories is possible. At its core, the strategic lane change algorithm is based on the maximization of the drivable distance without changing lanes while minimizing the number of necessary lane changes. In this respect a measure of urgency exists to rank the competing necessities (lane change versus maximization of driving distance), that correlates with a variety of variables including the remaining distance to the dead end, the distance to the end of the lane, the occupation of the ultimate target lane and the occupation of the intermediate target lane. Cooperative changes differ substantially from strategic lane changes as they have the sole purpose of helping other vehicles reach their desired lane. In the current implementation, whenever a vehicle is stationary in the lane and blocking subsequent drivers, it may change lane to clear a gap for vehicles behind it if this decision does not conflict with its strategic reasons. Tactical lane-changing refers to maneuvers in which a vehicle attempts to avoid following a slow leader. It requires the overtaking vehicle to strike a balance between the necessity of keeping the overtaking lane clear and the expected speed gains once the slow vehicle has been surpassed. The implementation of this procedure is strongly dependent on the overtaking rights of each vehicle and on the typology of lanes considered. Crucially for the simulation of realistic scenarios, legislation concerning overtaking speeds and rights varies amongst countries and has to be therefore considered carefully when defining the network's implementation. Finally, regulatory lane-changing takes place whenever a vehicle is occupying the designated overtaking lanes. Technically, the obligation to clear the overtaking lane could be framed as cooperative behavior described by cooperative lane-changing, however, cooperative lane-changing is highly customizable and the necessity for a strict implementation of user-independent regulated dynamics was met by the developers.

2.2.3 Intersection Dynamics

The final component of the simulation is the algorithm ruling intersection dynamics. As for the previous lane-changing, car-following, and pedestrian models, this section should be considered as a brief primer to the complexities of the actual implementation. The reader interested in a deep explanation may find it in [10]. We have already had the opportunity to consider the fundamental entities of SUMO scenarios; each road network consists of incoming and outgoing edges, where an edge represents a road with one or more lanes. Each lane possesses a unique identifier (id) which is derived from the edge identifier and the numerical index of the lane starting with 0 at the rightmost lane. The lanes of incoming edges are called incoming lanes, whereas the lanes of outgoing edges are known as outgoing lanes. When two edges intersect, we denote the intersection point as lane intersection. The purpose of this section is to clarify the structure of these junctions and the dynamics of the agents interacting with them. Within an intersection, lie so-called "internal lanes", which connect the incoming with the outgoing lanes. Once a vehicle reaches the intersection, it proceeds along these internal lanes just as it would on regular lanes within edges. At most intersections, including the one considered by this thesis work, vehicles wait at the end of their incoming lane at the border of the intersection until they can cross conflicting streams of traffic. However, SUMO also allows for more complex schemes in which left-turning vehicles can wait in the middle of the intersection. This is modeled by splitting internal lanes at the halting position and introducing an intersection within the intersection. Vehicles using these internal lanes always pass the entry link to the intersection and then wait at the internal intersection instead. The right-of-way computation for internal intersections follows the same principles as that of regular intersections. SUMO's policy in the computation of the intersection dynamics is to try to avoid collisions between agents. Nevertheless, the behavior of a class of vehicles at an intersection can be controlled via the specification of Junction Model Parameters. They include the possibility for a given vehicle to ignore other vehicles that have right-of-way with a given probability. This value also applies to the default pedestrian model. Also, it is possible to ignore other vehicles and pedestrians that have already entered a junction with a different given probability parameter:"JmIgnoreFoeProbability". Via careful specification of these parameters, it is possible to circumvent the non-colliding limitations imposed by the simulation to produce accidents. Chapter 5 will discuss how these parameters have been set and their impact on the obtained results.

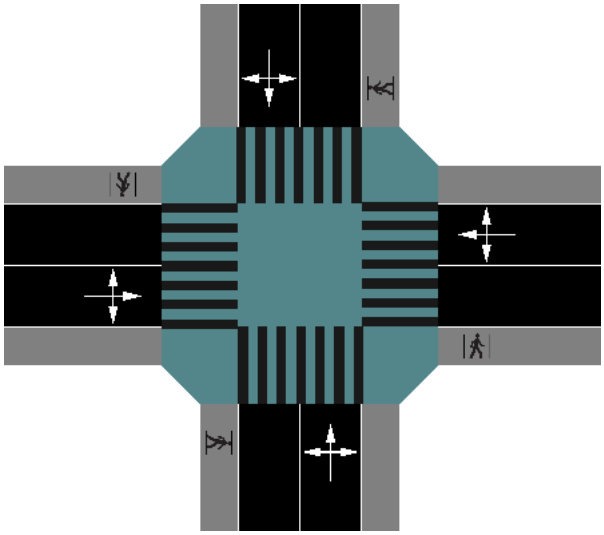


Figure 2.1. Graphical representation of the intersection at the core of the benchmark road network. In blue the intersection, in black the incoming vehicle-reserved lanes, and in gray the pedestrian sidewalks.

2.3 Proposed Network

As the aim of this thesis work is to present an effective framework for the generation of initial conditions leading to collision events in SUMO, the choice of an appropriate benchmark road network has been central. Ideally, an acceptable road network should abide by a series of requirements. The first is obvious, as it should be sufficiently expressive, that is, it should be capable of generating a variety of intra-vehicular and inter-agent collision events. A good testing benchmark should also be *computationally inexpensive* from a simulation viewpoint, as a great number of simulations have to be computed to generate a representative dataset, whilst being non-trivial. Under the limitations implied by these constraints, we propose the road network depicted in Figure 2.1. It consists of 4 edges incoming into a 4 way regulated intersection. Each edge has a length of 1000m from its start to the center of the intersection and contains 4 lanes. The external ones are sidewalks reserved for pedestrians and allow for movement in either direction, whereas the two central lanes follow the left-driving convention. Notably, the fact that each line is reserved for a single direction implies that no lane-changing behavior will be present. At the intersection, each vehicle-reserved lane can proceed onto each one of the three outgoing lanes belonging to the other edges. Pedestrians, on their part, can cross the intersection by traversing the pedestrian crossings that divide the edges from the intersection's center. During a routine simulation, we generate both pedestrian and vehicle demand and let the system evolve in time. Vehicles

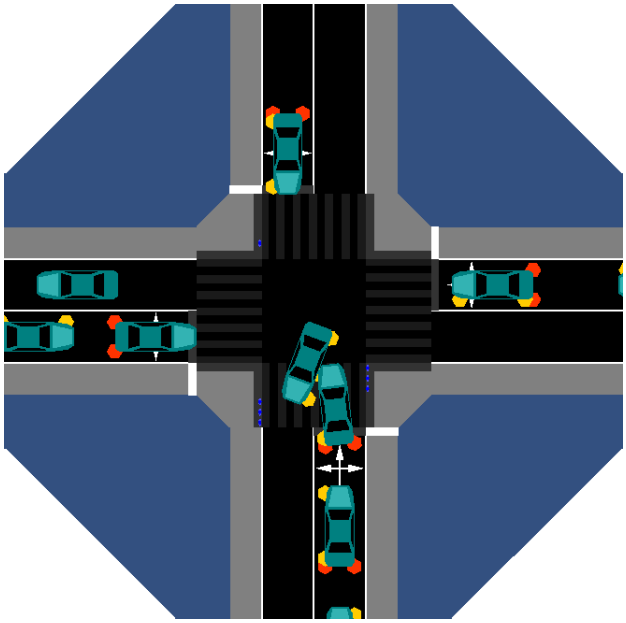


Figure 2.2. Collision event in the unfolding of a standard sumo simulation. The vehicle incoming from the bottom lane violates the right of way and collides with the vehicle already within the intersection. In navy blue the spectating pedestrians.

and pedestrians approach the four-way intersection in lines and await their turn to cross. A certain probability of ignoring the junction's right of way is given (as introduced in the previous section), consequently causing collisions. As the aim of this introductory Chapter is to provide an overview of SUMO and its main features, we refrain from discussing the implementation of vehicle and pedestrian demand employed in the simulation, for it will be extensively analyzed in Chapter 5.

Chapter 3

Classification

During the unfolding of a simulation, vehicles and pedestrians may be involved in several accidents. Pedestrians may collide while crossing a lane with an incoming vehicle that has been unable to break effectively and vehicles may crash into each other whenever the intersection's rights of way have been violated. To effectively provide a distinction between scenarios a classification system must be provided. This is the objective of the present chapter, where a gravity-based event classification scheme is proposed. Section 3.1 presents the relevant literature background, while Section 3.3 recounts the adaptation employed in the present work.

3.1 Measuring collision gravity

To correctly frame the problem of collision events between agents it is necessary to provide a common measure of gravity. In this work whenever we consider gravity we are implicitly referring to the damage experienced by either pedestrians or drivers, based on the typology of collision. For example, when considering the collision event between an incoming vehicle and a standing pedestrian, we refer to the gravity of the scenario by considering the pedestrian's point of view. Conversely, in vehicle-vehicle collisions we consider gravity from the perspective of drivers (or passengers). In this direction, the first step that needs to be taken is the definition of an objective measure of gravity. This is not as trivial a task as it may seem, for a person may have more coexisting injuries at the same time, all of different gravity. It is, therefore, necessary to first define a measure of gravity for the single injury, and then an aggregate one to consider the situation from a holistic point of view. The former issue has been a central point of contention for six decades ever since 1969 [28] when the development of what is today known as the Abbreviated Injury Scale (AIS) began within the Association for the Advancement of Automotive Medicine. In the past decades several modifications and adjustments

AIS-Code	Injury	Example
1	Minor	superficial laceration
2	Moderate	fractured sternum
3	Serious	open fracture of humerus
4	Severe	perforated trachea
5	Critical	ruptured liver with tissue loss
6	Maximum	total severance of aorta
9	No further specified	

Table 3.1. AIS-Codes in increasing gravity.

have been provided with major updates in 1976, 1980, 1985, 1990, 1998, 2005, 2008, and 2015. Crucially, the AIS represents the threat to life associated with the injury rather than the comprehensive assessment of the severity of the injury. Table 3.1 provides the denominations associated with the AIS, with exemplifying features that may give an approximate idea of the kind of lesion associated with each level. For example, a patient suffering from both a fractured sternum and a perforated trachea is said to be suffering from one AIS-Code 2 and one AIS-Code 4 lesion. Level 9 can be attributed whenever crucial information regarding the lesion is missing. Although reasonably effective in giving indications concerning individual injuries, the AIS code is not sufficient whenever a single gravity measure is necessary to evaluate a situation, as it is far too detailed to be of practical use. Particularly in the case of life-threatening injuries, we are not interested in a complete classification of the patient's injuries. On the contrary, we would like to assign just one global measure that characterizes the gravity of the situation. This can be achieved easily by considering the *maximum* AIS-Code assigned to a patient, thus defining the Maximum Abbreviated Injury Scale (MAIS). The MAIS code is the de facto standard employed by the traffic research literature to categorize injuries. It is constantly employed by regulatory bodies to provide insights on the overall safety of their road networks [11] and can be used to formulate probabilistic models of collisions of gravity. Contrary to the AIS-Code, the MAIS provides a unique number, measuring the immediate criticality of the assessed situation. In the following section, we will discuss how the MAIS can be used as a regression target to produce probabilistic models that correlate dynamical variables such as kinetic energy to the outcome of a collision.

3.2 Probabilistic Models

Intuition suggests speed to be the principal predictor of collision events. Experimental analysis supports this assumption, demonstrating that lower mean traffic

speeds in response to speed limit result in reduced likelihood of crashes. [1] The severity of crash outcomes has also been studied in correlation with the speed of the colliding parties. Interestingly, crash reconstruction research suggests that the estimated or measured impact speed of a vehicle is *not* an effective predictor of collision gravity in vehicle-vehicle collisions. Indeed, this is to be expected, as even identical vehicles can behave differently based on a variety of factors including their relative positions at collision time. The solution to this issue has been well known since the 1970s [23], when the idea of considering a vehicle's delta-v instead of its speed was proposed for the first time. In physical terms, the delta-v ΔV for a collision is the magnitude of the difference between a body's velocity after the collision and the one it had before the collision took place. In this field, it is customary to model the crash between two vehicles as an inelastic collision where only momentum is conserved. Vehicles are modeled as point masses in $\mathbb{R}^2$, so that given v_1, v_2 respectively the velocities of each vehicle before the impact and $m1, m2$ their masses, it is possible to require:

$$(m_1 + m_2)v^{after} = m_1 v_1 + m_2 v_2,$$

where v^{after} is the common velocity of the two bodies after the collision. To reduce the computational burden at a minimum, we select as the frame of reference the one implied by the orthonormal basis coherent with v_1. We can then write explicitly:

$$m_1 \begin{pmatrix} v_1 \\ 0 \end{pmatrix} + m_2 \begin{pmatrix} v_2 \cos(\phi) \\ v_2 \sin(\phi) \end{pmatrix} = (m_1 + m_2)v^{after}.$$

Where ϕ is the angle between the two vehicles. Via trivial manipulations it is possible to show that ΔV_1 and ΔV_2 satisfy:

$$\begin{aligned} \Delta V_1 &= \|v^{after} - v_1\|_2 = \frac{m_2}{m_1 + m_2}\sqrt{\langle v_1, v_2 \rangle} \\ \Delta V_2 &= \|v^{after} - v_2\|_2 = \frac{m_1}{m_1 + m_2}\sqrt{\langle v_1, v_2 \rangle}. \end{aligned} \tag{3.1}$$

Figure 3.1 conveys a graphical representation of an collision of the kind just described. The set of equations in 3.1 shows that ΔV can be computed for either vehicle in the collision. From now onward we refer to ΔV meaning ΔV_2. Although seemingly arbitrary this choice is inconsequential for the present work, as the vehicles considered in the simulations share the same mass. As researchers point out, empirical difficulties arise when trying to compute real-world delta-vs. However, in a simulated scenario, all the required variables are known with certainty, so that

[1] Exploration of vehicle impact speed – injury severity relationships for application in safer road design, Literature review of pedestrian fatality risk as a function of car impact speed

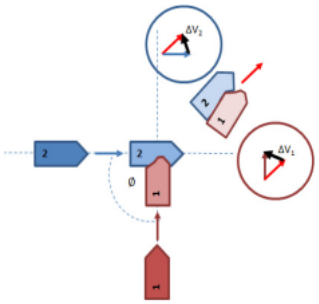


Figure 3.1. Schematic representation of the results in Equation(3.1). After the inelastic collision, the vehicles proceed forward as one with ΔV as in (3.1)

.

no uncertainty-induced error needs to be considered and the computations can be carried out with ease. Furthermore, it is important to recognize the limitations that the simplistic approach implies, as the delta-v of an individual crash depends on additional factors such as:

- relative masses of the vehicles,
- section of the target vehicle hit,
- vehicle construction materials,
- brake application and skidding,
- rotation induced momenta and
- post impact collisions.

Although significant, these factors do not hinder an effective analysis of collision scenarios between vehicles, as they all can be neglected except for one. Indeed, established literature [18, 23] employs ΔV as a predictor in a logistic regression problem where the target variable is the probability of experiencing MAIS3+ injuries for the colliding vehicle's driver. This, admittedly simplistic, approach has yielded considerable results and has been the starting point for a variety of subsequent models, that add to the covariate parameters from the previous list. This has been the case for the work proposed in [18], where the authors show that by categorizing collisions based on the relative positions of the colliding vehicles, probabilities of MAIS3+ injuries are obtained that sensibly fit empirical data. On the other hand, the study of vehicle-pedestrian collisions admits simpler analysis, as the physics of the collision in itself are different. Crucially, the large difference in masses between the bullet vehicle and the target pedestrian implies that collisions

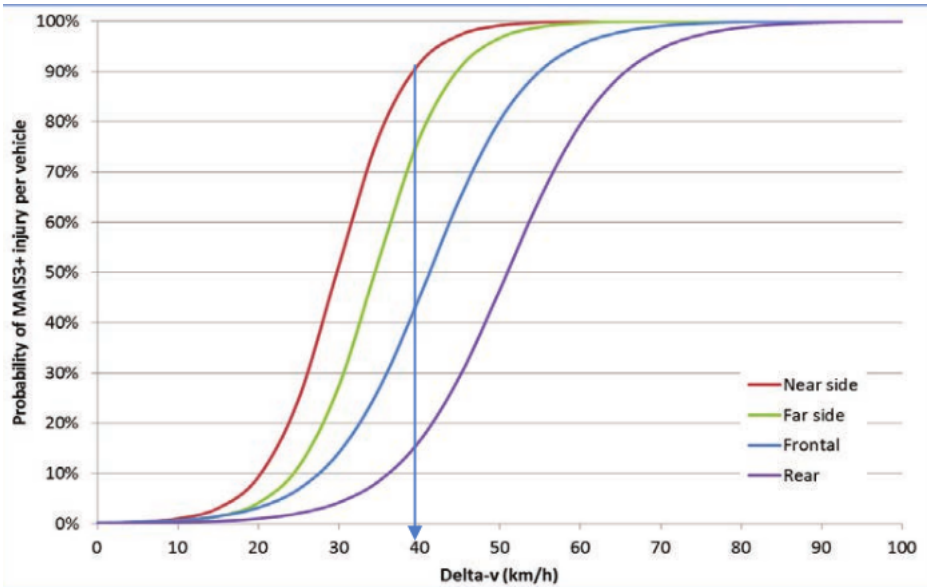


Figure 3.2. Lognormal curves with ΔV as predictor for MAIS3+ injury occurrence. The curves differentiate between near, far, front and rear end collisions. The image has been borrowed from [referenzaa]

can be modeled without the delta-v principle. Instead, the impact speed of the colliding vehicle is a sufficient predictor of gravity, and other factors such as the collision angle or the vehicle's construction materials can be ignored. [2] The final concept to be defined before discussing the resulting models is the idea of *critical impact speed*. The critical impact speed is the minimum bullet vehicle's speed that causes a MAIS3+ level injury to the target driver with a probability greater or equal to 10%. As the previous paragraph argued, speed is not a good predictor of impact gravity (in the MAIS3+ sense) for vehicle-vehicle collisions. However, for a specific set of assumptions detailed in the previous derivation, it is possible to develop a satisfactory fitting. In the following section, the necessity for such a metric will be discussed and analyzed in detail.

[2]Strictly speaking the statement is false, as different combinations of materials and geometries have been proven to be correlated with higher injury probabilities. Nevertheless, the action of safety regulations bodies has imposed strict constraints in the design of vehicles so that for real-world scenarios the assumption of independence can be tolerated

3.3 Proposed Classification Scheme

Although effective, the lognormal fitting provided in Figure 3.2 cannot be directly employed to derive a metric of gravity for the purposes of this work. The fundamental issue hindering its direct adoption is the fact that the distributions are continuous. This implies that the distinction mechanism is excessively fine, as collisions with ΔV differing by negligible amounts are mapped to different probabilities. Instead, we would like a more coarse classification scheme, still capable of retaining the general gravity of the situation, without focusing on negligible differences. To this extent, we make effective use of the lognormal fittings provided in Figure 3.2, by specifying for each type of collision three threshold values that coincide with selected probability levels. For example, assuming a vehicle-vehicle frontal collision, we propose to discretize the lognormal curve so that for values of $\Delta V \leq 7.0m/s$ the collision is classified as minor, with $7.0m/s < \Delta V \leq 12.5m/s$ as serious and finally with $\Delta V > 12.5m/s$ as grave. With this choice we are enforcing upper bounds to the MAIS3+ injury probability for each kind of collision. Indeed, a minor collision is an event having a MAIS3+ injury probability lower or equal to, for example, 10%, a serious one has an upper bound of 50%, and finally the term grave is reserved for the remaining values. Clearly, it is possible to customize and extend this scheme to every possible collision scenario generated within a SUMO simulation by selecting different probability thresholds for each possible collision event. Following the work in [18], we can distinguish between four possible events in vehicle-vehicle collisions based on the relative impact position of the two agents and one for vehicle-pedestrian collisions (as the relative positions of the incoming vehicle and the standing pedestrian can be neglected), thus giving rise to fifteen different classes. Furthermore, it is possible to extend this scheme to include emergency braking events that SUMO considers dangerous. In this direction it is possible to follow the same principles outlined in the present paragraph by subdividing the emergency braking event into three classes based on the velocity of the braking vehicle. Indeed, the higher the velocity of an incoming vehicle that has initiated an emergency braking maneuver, the higher the collision risk. Nevertheless, in the present work we refrain from implementing our classification technique to construct a representative dataset, because they can be easily generated by manually setting each vehicle's velocity so that the following vehicle has substantially higher speed than the leading one.

3.3.1 SUMO's implementation

As we have hinted in previous paragraphs, real-world estimation of the predictors needed by the lognormal fitting is, in general, a complex task. For example, in collisions between vehicles the assumptions underpinning the use of the ΔV as a

Agents	Typology	Low	Medium	High
Vehicle-Vehicle	Frontal	[0, 7.0]	(7.0, 12.5]	(12.5, ∞)
	Lateral driver	[0, 5.5]	(5.5, 8.0]	(8.0, ∞)
	Lateral	[0, 6.0]	(6.0, 11.0]	(11.0, ∞)
	Rear	[0, 10.0]	(10.0, 15.0]	(15.0, ∞)
Pedestrian	All	[0, 8.0]	(8.0 12.5]	(12.5 ∞)

Table 3.2. Tabular representation of the classification scheme proposed in the previous section. The numerical intervals correspond to the discretization employed in our simulations.

regression parameter can be easily violated. Furthermore, estimating the ΔV is in itself an arduous task, as it requires a ballistic analysis of the collision event. On the contrary, the task is considerably simpler for events between pedestrians and vehicles, as in this case the problem can be reduced to the estimation of the colliding vehicle's speed. In the context of the simulations computed by SUMO, however, these difficulties subside because we can access detailed data regarding the position and velocity of every agent within the simulation for every dt. The estimation of ΔV becomes then a trivial geometrical problem and the classification that we have advocated for can be employed effectively. Motivated by the straightforward implementation of the proposed scheme, we have analyzed the classification results of 9902 events based on the road network presented in the previous Chapter 2 and on randomly generated demand for pedestrians and vehicles. Based on the selected SUMO stochastic parameters, which will be discussed in depth in Chapter 5, it is possible to appreciate how the proposed road network favors a subset of specific typologies of events. Indeed, among the possible plethora of collisions that the proposed classification system admits, the benchmark devised by the presented research has produced events belonging to just 6 classes (not considering the emergency braking). This is to be expected, as it is a direct consequence of the combination of parameters and geometry of the benchmark. Moreover, this phenomenon is akin to what can be observed in real-world situations, where it is possible to appreciate a clustering of similar accidents depending on the contingent characteristics of the road network.

Chapter 4

Generative Adversarial Networks

This Chapter describes the learning methodologies devised in the development of the present work. The discussion begins with an introductory primer concerning the field of Machine Learning in Section 4.1. We then move to characterize the family of algorithms known as Artificial Neural Networks in Section 4.2, analyzing their structure and learning methods. Finally, Section 4.3 introduces the theory of Generative Artificial Networks and details how they have been implemented in the context of this thesis work.

4.1 A brief introduction to Machine Learning

The term Machine Learning was first used in 1959 by IBM engineer Arthur Samuel to denote statistical algorithms capable of learning from data. In its essence, Machine Learning postulates the existence of a random variable X, whose realizations $x^{i=1,\dots,N}$ (the data), constitute the observable states of the system of interest. Coupled to each realization $x^{i=1,\dots,N}$, a response variable $y^{i=1,\dots,N}$, realization of the random variable Y may be present. Based on the presence and the nature of Y, we can distinguish three scenarios:

- **Supervised Learning**, where the computer is presented with inputs $x^{i=1,\dots,N}$ and associated outputs y^i and the goal is to learn a map $f : X \to Y$.

- **Reinforcement Learning**, where a computer program interacts with a dynamical system. In this context, the model tries to maximize a feedback reward provided by the system itself.

- **Unsupervised Learning**, where realizations $x^{i=1,\dots,N}$ of the random variable X are known and characterizations of interest are sought.

In the first two instances, the ML model aims to learn the mapping $f : X \to Y$ between the space of the data X and the target space Y that minimizes a given error. The third scenario differs substantially, as the aim of the ML model becomes to retrieve regularities from data, rather than map fitting. In the next subsections, we will briefly describe the main features of each approach, to then focus on a family of learning algorithms known as Deep Neural Networks,

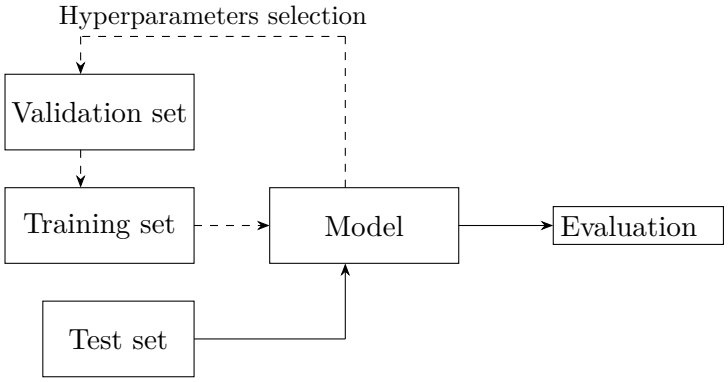


Figure 4.1. Development pipeline of a supervised learning architecture. Hyperparameters for the selected model are specified and the model itself is trained on the training set and evaluated on the validation set. Once a satisfactory set of hyperparameters is found, the model is evaluated on the test set.

4.1.1 Supervised Learning

Supervised learning aims to fit mathematical models to minimize an error norm on data that contains both the input realizations drawn from X and the response variable Y. The set is known as *training set*. In the mathematical model, each training sample is represented by a vector known as *feature vector*, and the training data is represented as a matrix, whose rows are the individual data points and the columns are generally known as features. Through iterative parameter optimization of an objective function, supervised learning algorithms learn a function that is used to predict the associated outputs. Ideally, an optimal function allows the algorithm to correctly determine the output for inputs that were not a part of the training data. Within the field of Supervised learning, it is possible to differentiate between those Y having continuous and discrete support. In the first scenario, we define the problem as classification, in the latter we call it regression. A classical example of classification can be found in deciding whether or not an email belongs

to the spam folder or not. Conversely, a standard example of regression learning may be finding the mapping between a London borough and the corresponding house prices. The development of a standard supervised learning pipeline starts with a clear definition of the task to be addressed: whether to consider the classification of handwritten text or the sentiment analysis of tweets, clearly stating the problem of interest is crucial. Once the task has been defined, it is possible to analyze the requirements that the dataset must fulfill to address the task and train the model. The problem of generating a representative training set is central in machine learning, as corrupt, missing, or incoherent data can disrupt the algorithms' learning process. Once the dataset has been constructed, the input feature representation of the mapping is discussed. Statistical analysis concerning correlated features, dimensionality reduction, and feature engineering are methods generally employed to define the feature vectors to be used in the learning process. Based on the complexity of the task and the available data, the structure of the learned mapping is determined. In this direction, a plethora of possible algorithms are available, such as classification trees, support-vector machines, deep neural networks, random forests, and many more. Crucially, the functional space implicitly spanned by the chosen method must attain similar complexity to the target mapping. Otherwise, there is no hope for the learning algorithm to perform satisfactorily on the specified task. To effectively train the model, an error metric must be specified and a hyperparameter selection strategy implemented such as grid search. The final step once the algorithm has been trained is to evaluate its performance on a test set that is completely independent of the training set. Figure 4.1 conveys a schematized version of this procedure. Supervised learning has been used to effectively solve a variety of different problems, ranging from bioinformatics to database marketing, and is an exceptional tool whenever classification or regression are required.

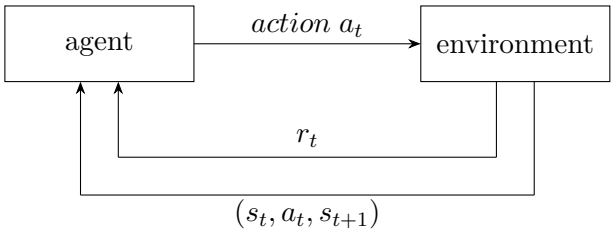


Figure 4.2. Schematized representation of the reinforcement learning paradigm. An agent interacts with an environment via actions a_t and receives the tuple (s_t, a_t, s_{t+1}) and a reward r_t of being in the previous state s_t.

4.1.2 Reinforcement Learning

In reinforcement learning an agent interacts with an environment through actions to obtain rewards. The interactions between the agent and the environment are generally divided into periods of uniform time called episodes. Upon completion of each episode, the agent modifies its behavior based on the rewards obtained by its previous strategy and another episode starts until either the maximum number of allowed episodes is reached or the agent learns the optimal strategy. From a mathematical point of view, reinforcement learning interfaces with the field of optimal control, concerned with how an intelligent agent ought to take actions to maximize a cumulative reward. Due to its generality, reinforcement learning is studied by many disciplines, including game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, and statistics. Interestingly, in the operations research literature reinforcement learning is known as *approximate dynamic programming*, as it is considered an approximate solution method to the Bellman optimality equation. Basic applications of the reinforcement learning paradigm model the system of interest as a Markov decision process, where a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition probability $\mathbb{P}_a(s, s^{'}) = \mathbb{P}(S_{t+1} = s^{'}|S_t = s, A_t = a)$ and an instantaneous reward $R_a(s^{'}, s)$, obtained after the transition from s to s'. In this framework, an agent interacts with the Markov process at every discrete time t by receiving the current state s_t and the reward from the previous transition r_t. It then chooses an action $A_t \in \mathcal{A}$. Once the action has been sent to the environment, a new state s_{t+1} is reached and a consequent reward r_{t+1} associated with the tuple (s_t, A_t, s_{t+1}) obtained. Figure 4.1.1 schematizes this iterative process. The objective of the reinforcement learning algorithm is therefore to learn a policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, such that $\pi(s, a) = \mathbb{P}(A_t = a|s_t = s)$ that maximizes the expected cumulative reward. During the past decade, the reinforcement learning paradigm has collected an outstanding number of successes in real-world applications, including optimal option pricing for financial derivatives and superhuman skill in complex strategic games such as chess or go. Its crucial advantage when compared with analytical-based methods is the fact that no explicit model of the system is needed at learning time. The agent is completely agnostic to the hidden dynamics of the system, that is embedded in the simulation. Therefore, the only impacting aspect of the environment experienced by the agent is (s_t, A_t, s_{t+1}, r_t), which can be used to train learning algorithms that implement the policy π to recognize the actual value of being in a given state s_t. In this direction, the reinforcement learning problem behaves somewhat like a noisy version of the supervised learning paradigm, with the fundamental difference being that the reward r_t is but a perturbed estimate of the actual value of being on a given state s_t.

4.1.3 Unsupervised Learning

This brief overview of the main areas in machine learning concludes with unsupervised learning. In unsupervised learning, in contrast with both reinforcement and supervised learning, algorithms learn patterns exclusively from unlabeled data. To this extent, a further subdivision is possible between generative and clustering models. Cluster analysis is the task of grouping a set of objects in such a way that objects belonging to the same class (i.e. cluster) share similar characteristics specified by the analyst. The notion of clustering does not possess a specific intrinsic definition, as different application cases imply diverse unifying features. Indeed, this is reflected in the vast amount of different clustering algorithms, each giving a different definition of cluster and its properties. For example, in hierarchical clustering data points are clustered by a specified distance connectivity (generally Euclidean). Moreover, centroid models such as the k-means algorithm represent each cluster by a single mean vector (the centroid), and each point is said to be part of the cluster specified by the nearest centroid. Model-based clustering such as Gaussian mixture models, on the other hand, maps the data with a fixed number of Gaussian distributions initialized at random and then trained using the expectation-minimization algorithm. Finally, density-based clustering methods, such as DBSCAN, only connect points that satisfy a given density criterion, in the original variant defined as a minimum number of other data points (neighbors) within a given radius. A cluster consists then of all the density-connected objects (that can form clusters of arbitrary shape, in contrast with Gaussian mixture models) plus all the objects that are within this object's range. Once the clustering has been completed, some intrinsic and extrinsic evaluation measures exist to compare the quality of a given clustering. Intrinsic evaluations provide a metric computed within each cluster, whether external scores compute comparisons between clusters. Interestingly, internal evaluation measures suffer from the issue that they represent a clustering objective. For example, given a specific intrinsic score to be minimized (or, equivalently, maximized), one would like to develop an efficient algorithm to compute a clustering that minimizes said score. Similarly, extrinsic evaluation scores suffer from similar issues, as we would like to compute a clustering that assigns each data point to the "correct" cluster. The issue is that if we had had the "correct" label for each point then we would not have employed unsupervised learning algorithms. Therefore neither extracluster nor intracluster evaluations can ultimately judge the actual quality of a clustering, and human evaluation, although highly subjective, is still needed.
Generative models, on the other hand, aim at learning a representation of the underlying probability distribution that generates the dataset X. Formally, a generative model is a model of the conditional probability of a given observable X

given Y: $P(X|Y = y)$. In this direction, the terminology is quite adequate because once a representation of the conditional probability is known, it is possible to draw realizations from the random variable $X|Y$, i.e. to generate samples from the distribution. It is important to notice that the problem of learning a target distribution X is, in general, of greater complexity than classification or regression as defined by supervised learning. This is caused by several different factors, the most prominent of which is known as the curse of dimensionality. The expression curse of dimensionality, in this context, denotes the breaking down of traditional statistical methods caused by the topology of high dimensional spaces. Indeed, traditional generative methods such as Gaussian mixture modeling or Generalized Linear Models have proven unable to provide adequate results when applied to high dimensional problems such as image generation. Therefore, new methodologies have been developed during the last decade to effectively solve the problem of data generation in arbitrary spaces. Those methodologies rely on deep neural networks trained via backpropagation to provide sensible estimates of the probability distribution generating the data. We will discuss in detail artificial neural networks and deep learning in the following Section 4.2 to present the methodology employed by this thesis for data generation. The success of deep neural networks for data generation is ubiquitous: from custom image generation as provided by DallE-4, to Large Language Models such as ChatGPT 4o, deep learning has proven to be an exceptional generative framework capable of effectively dealing with the problems posed by high-dimensional data spaces.

4.2 Artificial neural networks

Amongst the plethora of machine learning algorithms, artificial neural networks occupy a prominent stage. Their lineage can be traced back to the early 1960s, when Frank Rosenblatt, in his book [27] outlined the fundamental aspects of the field that would be known as deep learning. In his seminal work, Rosenblatt presents all the components of a working feedforward artificial neural network. Inspired by simplistic models of the brain, he sought to imitate its dynamics via a directed acyclic graph (DAG): a peculiar type of graph where it is impossible to end up on a starting vertex by following the arcs that leave it. Following his brain analogy he calls each node of the network *neuron* so that each arc of the network represents a synopsis in his simplified model. Rosenblatt denotes this particular species of feedforward artificial neural network multilayer perceptron. In his definition, multilayer indicates that neurons can be organized in successive layers, called hidden layers, based on their incoming connections, while perceptron highlights his neurological analogy. Although seminal, Rosenblatt's work did not provide a general working learning algorithm for supervised feedforward multilayer

perceptrons. Indeed, the first working stochastic gradient descent algorithm for multilayer perceptrons was published in 1967 by Shun'ichi Amari [2]. Shun'ichi's contribution demonstrated the feasibility of optimization-based training and constitutes a fundamental milestone in the history of artificial neural networks. Since 1967, a wide range of different architectures have been proposed, classified as either feedforward or recurrent based on the presence of cycles in the underlying network model. The purpose of this section is to introduce the main features defining feedforward artificial neural networks from a mathematical perspective. In its mathematical formulation, the theory of artificial neural networks requires elements of Linear Algebra, Real Analysis, Probability, and Optimization theory that are not covered in this brief introduction. Indeed, the reader should consider this section to serve as necessary background to understand the Generative Adversarial Network developed in this thesis work.

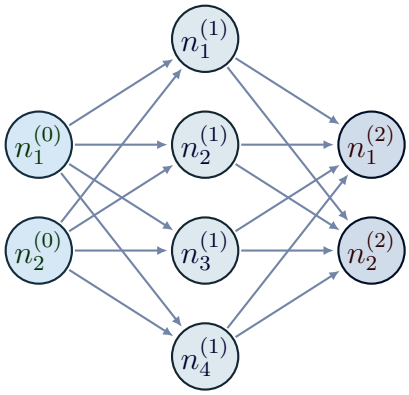


Figure 4.3. Graphical representation of Rosenblatt's first multilayer perceptron. From the left: Input layer consisting of two neurons, the hidden layer of 4 neurons, and the final output layer consisting of 2 neurons. Shun'ichi's work surpassed Rosenblatt's proposed training algorithm allowing the development of more general architectures.

4.2.1 Architecture

This paragraph provides a formal mathematical definition of a feedforward artificial neural network employing concepts from graph theory. In doing so we identify the network with its computational and topological structure, therefore enabling the definition of the training dynamics object of the following paragraphs.

Definition 4.2.1. An artificial feedforward neural network is a composite parametric function $f_w : \mathbb{R}^n \to \mathbb{R}^m$, whose composition structure can be represented as a weighted directed acyclic graph $G = (\mathcal{W}, \mathcal{N})$ whose nodes, called *neurons*,

represent the action of a function $\sigma_i : \mathbb{R} \to \mathbb{R}$ called *activation function*.

Remark 4.2.2. We denote with $\mathcal{W}_{-i}$ the set of indexes of the nodes entering n_i.

Remark 4.2.3. Definition 4.2.1 is of notable generality, as it encompasses arbitrary network topologies. Most applications employ strict compartmentalization between inputs and outputs through hidden layers, as shown in Figure 4.2.

Remark 4.2.4. In Definition 4.2.1 the neurons represent the functions that constitute the composition, whereas the learning parameters w_i are represented by the connections between neurons.

Although any activation function σ_i may be chosen for any node n_i of the network, applications privilege a variety of well-behaving functions, the most important of which are the objects of the following definitions.

Definition 4.2.5. We call Rectified Linear Unit the function $f : \mathbb{R} \to [0, \infty)$ of the form:

$$f = \max\{0, x\}. \tag{4.1}$$

Definition 4.2.6. We call Leaky Rectified Linear Unit the function $f : \mathbb{R} \to \mathbb{R}$ of the form:

$$f = \max\{0, x\} + \alpha \min\{0, x\} \tag{4.2}$$

Definition 4.2.7. We call Sigmoid the function $f : \mathbb{R} \to [0,1]$ of the form:

$$f = \frac{1}{1 + e^{-x}} \tag{4.3}$$

Definition 4.2.8. We call Hyperbolic tangent the function $f : \mathbb{R} \to (-1,1)$ of the form:

$$f = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{4.4}$$

Having completed an initial characterization of the defining features of an artificial feedforward neural network, it is now possible to understand its behavior in terms of its computational graph. First, input data $x \in \mathbb{R}^n$ is fed to the input neurons of the network, i.e. neurons that do not have any incoming connection. Let us then consider a generic neuron $n_i \in \mathcal{N}$, equipped with an activation function σ_i. During a forward passage, the selected neuron receives as input the weighted sum of the outputs of his incoming neighbors:

$$\sum_{j \in \mathcal{W}_{-i}} w_j \sigma_j(n_j),$$

and outputs the value of its activation function computed in the sum:

$$\sum_{j \in \mathcal{W}_{-i}} w_j \sigma_j(n_j) \to \sigma_i \left(\sum_{j \in \mathcal{W}_{-i}} w_j \sigma_j(n_j) \right).$$

This process is repeated for each non-input node in the network until the last node is reached. Remarkably, the fact that the network is defined by a DAG implies that this procedure is indeed well-defined, as no infinite loops are possible. Figure 4.4 describes graphically this process. It is important to comprehend that the choice of activation functions is crucial as it implicitly defines the functional space to which the resulting function belongs. For instance, if we want to choose exclusively identity functions such that $\sigma_i(x) = x, \forall n_i \in \mathcal{N}$, we would be limiting our network to compositions of identities, therefore forfeiting non-linearities. This important fact shall be made clear by the following example.

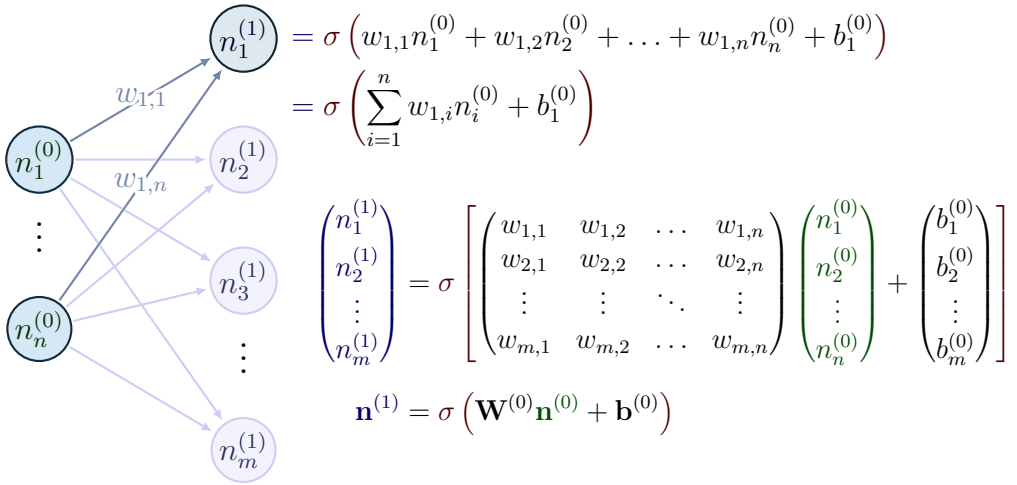


Figure 4.4. Mathematical representation of a hidden layer's output: each component x_i of the input vector x is multiplied by the corresponding weight $w_{i,j}$ that represents the arc between neurons i and j. The weights denoted by the vector b represent bias weights: nodes having no incoming connection that are not associated with inputs.

A linear example Let us consider the simplest possible neural architecture consisting of one input node and one output node. In this scenario, only one weight is present, as only one arch is present. The resulting approximating function will therefore take the form:

$$f = \sigma_i(w_i x).$$

If we take σ_i to be the identity over $\mathbb{R}$ we see that the function f belongs to the set of linear functions

$$f = w_i x.$$

Furthermore, using our formalism it is trivial to add another input node representing a unitary input. By doing so we refer to the case depicted in Figure 4.2.1, with two weights that expand the functional space to comprehend all affine functions from $\mathbb{R}$ to $\mathbb{R}$. Formally, this second weight is known as *bias*, as it corresponds to a neuron having no incoming connection and constant value 1.

$$f = w_1 x + w_2.$$

Until this point, we have limited the exposition to the definition and description of a neural network's structure and forward passage. It has been shown how the choice of activation functions impacts the functional space to which the approximation belongs. However, we have not yet mentioned how much approximators can be trained to minimize some kind of error metric on a given data set. This is the object of the following section.

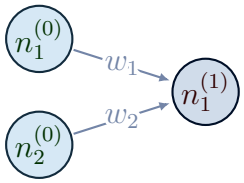


Figure 4.5. Graphical representation of the neural network developed in the example. Two input neurons $n_1^{(0)}$ and $n_2^{(0)}$ converge to the output node $n_1^{(1)}$. The first input node $n_1^{(0)}$ represent the actual data, while $n_2^{(0)}$ represent a constant bias of magnitude 1. With their combination with the identity activation function, we obtain an affine function.

4.2.2 Deep Learning

The term deep learning was proposed in 1986 by Rina Dechter [6] to describe the algorithms used to train deep neural networks. The adjective "deep" refers to the use of multiple layers in the network through which the input data is transformed. Formally, deep learning systems can be identified by a metric known as credit assignment path (CAP) depth. The CAP describes the causal connections between inputs and outputs. Feedforward neural networks have CAPs that are equal to the number of hidden layers plus one (the output layer) because their computational diagram is a DAG. There is no universally accepted CAP number to discern between shallow and deep learning; however general consensus of the

research community defines CAP equal to two as the threshold between the two. In this section, we will focus on the quintessential deep learning algorithm used to train feedforward deep neural networks known as *backpropagation*. To this aim, we will begin by placing the problem within the field of supervised learning by defining a dataset within the realm of Probability Theory and then introduce the stochastic gradient descent method used to update the network's parameters.

Definition 4.2.9. We call *dataset* $D = \{(x_1, y_1), ..., (x_i, y_i)\}$ the union of a collection of realizations x_i drawn from a random variable X, known as *instances*, with realizations y_i drawn from a random variable Y known as *targets*.

Intuitively, the objective of our machine learning problem is to find a mapping $f_w : X \to Y$ that associates to each instance drawn from the random variable X a value y that is the best possible w.r.t. some kind of error, hereby known as loss function. We will begin by giving the definition of a loss function and then proceed to understand how it can be used to train a feedforward network.

Definition 4.2.10. A loss function $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ is a function that quantifies the difference between two or more values.

The choice of the correct loss function is crucial, as it completely determines the mapping learned by the feedforward network. A variety of losses are available based on the considered task:

Definition 4.2.11. The *Mean Absolute Error*, MAE is the average of the $\mathbb{R}^1$ Euclidean distance between the network's outputs $f(x|w)$ and the targets y.

$$\text{MAE} = \mathbb{E}\Big[\Big\|y - f_w(x)\Big\|_1\Big] = \frac{1}{n}\sum_i \Big|f_w(x_i) - y_i\Big|.$$

Definition 4.2.12. The *Mean Squared Error*, MSE is the average of the $\mathbb{R}^2$ euclidean distance between the network's outputs $f(x|w)$ and the targets y.

$$\text{MSE} = \mathbb{E}\Big[\Big\|y - f_w(x)\Big\|_2^2\Big] = \frac{1}{n}\sum_i \Big(f_w(x_i) - y_i\Big)^2.$$

Both MAE and MSE are well-known losses used in regression problems. For classification purposes, the Categorical Cross Entropy Loss is generally used.

Definition 4.2.13. The Categorical Cross Entropy loss is the cross entropy between two probability distributions q and f_w sharing the same support $\mathcal{S}$:

$$H(q, f) = -\mathbb{E}\Big[log(f_w(x))\Big]_{q(y|x)} = -\sum_{\mathcal{S}} q(y|x) log(f_w(x))$$

Notably, by training a network with the Categorical Cross Entropy loss we are trying to learn the probability distribution $q(y|x)$, that is the conditional probability distribution of of having output y given input x. In fact, the Categorical Cross Entropy loss is a direct measure of the difference between two probability distributions. Unfortunately, in practice we do not know $q(x, y)$, therefore even the direct computation of the loss in this form is an intractable problem. In the following sections, we will see in detail how such computations can be carried out effectively to train a feedforward neural network to learn probability distributions.

4.2.3 Bias-Variance Tradeoff

Before delving into the intricacies of training algorithms for feedforward artificial neural networks, it is important to understand the fundamental tradeoff that characterizes machine learning models. In general, as the number of learnable parameters in a model increases it becomes more flexible being capable of learning complex mappings to minimize losses on the training data set. The model is therefore said to have achieved lower *bias*. However, this increased flexibility tends to cause higher *variance* to the model fit parameters whenever new data is added to the training set. In this context, it is possible to enunciate the bias-variance problem as the dichotomy that arises whenever we try to minimize both variance and bias that prevent supervised learning algorithms from generalizing effectively beyond the data used for training. In the following, we will define the expected generalization error for a generic machine learning model and then proceed to show an example of bias-variance decomposition. This is foundational material to support the architectural decisions made in defining the model proposed by this thesis work.

Definition 4.2.14. The *expected generalization error* E_f with respect to L is the expected value of the loss function l for a function f_w over all possible values x, y:

$$E_f^l = \mathbb{E}_{(x,y)\sim P(x,y)}\Big[\mathcal{L}(y, f_w(x))\Big].$$

In practice, E_f is approximated by computing the sample mean of the loss over a given test set. In this direction we would like E_f^l to be as small as possible. The following theorem shows how, even in the simple case of regression with Mean Squared Error loss, an irreducible error is unavoidable.

Theorem 4.2.15. *Consider a supervised learning problem where $y = \hat{f}(x) + \varepsilon$, where ε is a random variable of mean zero and variance σ^2. Then, the expected generalization error of the approximating function f trained in dataset D with respect to the MSE loss can be decomposed as follows:*

$$E_f^{MSE} = \mathbb{E}\Big[\big\|y - f(x; D)\big\|_2^2\Big] = \Big(Bias_D\big[f(x; D)\big]\Big)^2 + Var_D\big[f(x; D)\big] + \sigma^2 \quad (4.5)$$

Where:

$$Bias_D\Big[f(x;D)\Big] = \mathbb{E}_D\Big[f(x;D) - \hat{f}(x)\Big] = \mathbb{E}_D\Big[f(x;D)\Big] - \mathbb{E}_{y|x}[y(x)],$$

and

$$\sigma^2 = \mathbb{E}_y\Big[(y - \hat{f}(x))^2\Big].$$

Proof. The proof is a straightforward expansion of the expected generalization error $E_f^{MSE} = \mathbb{E}\Big[\big\|y - f(x;D)\big\|_2^2\Big]$. For the sake of notational simplicity, in the following proof whenever we write f we mean $f(x;D)$.

$$\mathbb{E}\Big[\big\|y - f\big\|_2^2\Big] = \mathbb{E}\Big[y^2 - 2yf + f^2\Big] = \mathbb{E}[y^2] - 2\mathbb{E}[yf] + \mathbb{E}[f^2]$$

We then reason separately for each term:

$$\begin{aligned}\mathbb{E}[y^2] &= \mathbb{E}\Big[(\hat{f} + \varepsilon)\Big] \\ &= \mathbb{E}[\hat{f}^2] + 2\mathbb{E}[f\varepsilon] + \mathbb{E}[\varepsilon^2] \\ &= \hat{f}^2 + 2\hat{f}\mathbb{E}[\varepsilon] + \mathbb{E}[\varepsilon^2] \\ &= \hat{f}^2 + \sigma^2\end{aligned}$$

Then for $\mathbb{E}[yf]$:

$$\begin{aligned}\mathbb{E}[yf] &= \mathbb{E}[(\hat{f}f + \varepsilon f)] \\ &= \hat{f}\mathbb{E}[f] + \mathbb{E}[\varepsilon]\mho \\ &= \hat{f}\mathbb{E}[f].\end{aligned}$$

Finally, putting it all together:

$$\mathbb{E}\Big[\big\|y - f\big\|_2^2\Big] = \sigma^2 + \hat{f} - 2\hat{f}\mathbb{E}[f] + \mathrm{Var}[f] + \mathbb{E}[f]^2 = \sigma^2 + \mathrm{Var}[f] + \Big(\mathbb{E}[f] - \hat{f}\Big)^2,$$

thus proving the statement. ■

Theorem 4.2.15 explicitly conveys the tradeoff between bias and variance in the context of supervised learning. Indeed, by augmenting the number of learnable parameters in the model, the bias term tends to zero. However, the variance term increases, thus limiting the generalization capabilities of the model. Crucially, even if a zero variance zero bias model was possible, the minimal generalization error would still be nonzero and coincide with ε's variance. For this reason σ^2 is called *irreducible error*. Therefore, when developing a learning architecture, it is paramount to try and understand the complexity of the learning task to actually test models that strike an effective balance in the bias-variance tradeoff. In the following section we will show how considerations of this kind have shaped the development process of the generative adversarial network presented in this thesis.

4.2.4 Loss Optimization

Having defined the general architecture of feedforward artificial neural networks, we shall now consider the problem of updating the parameters $w \in \mathcal{W}$ to minimize the expected generalization loss as presented in the previous paragraph. In this context, the fundamental technique employed to minimize E_f^l is called *Stochastic Gradient Descent* SGD. To be effectively applicable, SGD requires the estimation of the loss's gradients, which can be obtained via a process widely known as *back-propagation*. This brief exposition will focus on the standard machine learning scenario where the output of the function to differentiate (i.e. the loss) L is a scalar and we are interested in its dependence on the set of learnable parameters. By iteratively updating those parameters to minimize the observed loss, we can therefore produce models that learn the underlying mapping between instances and target variables.

Back-propagation The core property required in Definition 4.2.1 is the composition structure of the activation functions that constitute the neural network. In this foundational exposition we have required this structure to be isomorphic to a Directed Acyclic Graph, so that given a chosen neuron in the network, a random walk starting in it would always end up, in a finite number of steps, on the output nodes. This property is crucial as it allows the use of the celebrated chain rule to compute the gradient of the loss function with respect to the trainable weights w. Indeed, given a composite scalar function of the form discussed above $J : \mathbb{R}^n \to \mathbb{R}$, with $g_\theta : \mathbb{R}^m \to \mathbb{R}^n$ so that $J(g_\theta) : \mathbb{R}^m \to \mathbb{R}$, we can decompose its gradient using the following equality:

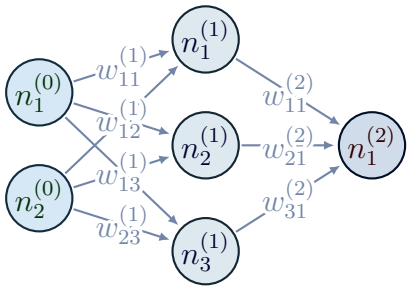
$$\nabla_\theta J(g_\theta) = \frac{\partial g}{\partial \theta} \nabla_{g_\theta} J(g_\theta).$$

Naturally, the equality can be extended also for vector functions with the aid of Jacobian matrices of appropriate dimensions. In Figure 4.6 we consider the case of a simple network consisting of two input neurons, one hidden layer of 3 neurons, and one output neuron, and show the dependence on infinitesimal variations of the learnable parameters of the final approximating function. For the trivial case of activation functions coinciding with the identity $\sigma_i(x) = x$, an explicit computation is straightforward given the functional form:

$$f_w(x) = \left(\sum_{j=1}^{3} w_{j1}^{(2)} \left(\sum_{i=1}^{2} w_{ij}^{(1)} x_i \right) \right),$$

for the last set of weights, it holds:

$$\frac{\partial f_w(x)}{\partial w_{j1}^{(2)}} = \frac{\partial}{\partial w_{j1}^{(2)}} \left(\sum_{j=1}^{3} w_{j1}^{(2)} \left(\sum_{i=1}^{2} w_{ij}^{(1)} x_i \right) \right) = \left(\sum_{j=1}^{2} w_{ij}^{(1)} x_i \right),$$



$$f_w(x) = \sigma_2\left(\sum_{j=1}^{3} w_{j1}^{(2)}\sigma_1\left(\sum_{i=1}^{2} w_{ij}^{(1)}x_i\right)\right),$$
$$\frac{\partial f_w(x)}{\partial w_{11}^{(1)}} = \frac{\partial f_w(x)}{\partial \sigma_2}\frac{\partial \sigma_2}{\partial \sigma_1}\frac{\partial \sigma_1}{\partial w_{11}^{(1)}},$$
$$\frac{\partial f_w(x)}{\partial w_{i1}^{(1)}} = \frac{\partial f_w(x)}{\partial \sigma_2}\frac{\partial \sigma_2}{\partial \sigma_1}\frac{\partial \sigma_1}{\partial w_{i1}^{(1)}}$$

Figure 4.6. Symbolic computation of the network's gradient via the recursive use of back-propagation.

whereas for the first layer, we have:

$$\frac{\partial f_w(x)}{\partial w_{ij}^{(1)}} = \frac{\partial}{\partial w_{ij}^{(1)}}\left(\sum_{j=1}^{3} w_{j1}^{(2)}\left(\sum_{i=1}^{2} w_{ij}^{(1)}x_j\right)\right) = w_{j1}^{(2)}x_i.$$

The resulting gradient lives in $\mathbb{R}^9$ and we will see how it can be used to compute the gradient of a given loss $\mathcal{L}$. From an algorithmic point of view, it can be easily proved that the back-propagation algorithm has optimal computational complexity, in the sense that there is no competing algorithm that can compute the gradient faster (in the $O(\dot{})$ sense) [12]. The fundamental property that renders back-propagation applicable on large-scale networks is the fact that partial derivatives can be decomposed recursively by taking into consideration the activation of the neural network's neurons. This is performed by specialized software such as Pytorch or Tensorflow [26, 1], where back-propagation is used to compute gradients up to machine precision.

Stochastic Gradient Descent Back-propagation is an optimal algorithm for the recursive computation of a function's gradient. However, it does not give any indication concerning the magnitude of weight updates to be performed in order to minimize some kind of loss $\mathcal{L}$. To this extent, the family of *gradient descent* algorithms is the seminal class of iterative methods employed to reach the minima of functions. The most basic element of this family is known as the gradient descent algorithm, which in the case of a deep neural network involves iteratively updating the weights vector w with a small step in the direction of the gradient of the loss $\mathcal{L}$. In supervised learning, this means computing the images of the training inputs x_i through f_w to obtain the training loss $\mathcal{L}(y, f_w(x))$. Through back-propagation, the gradient of the loss with respect to the weights w is then computed, and the weights are updated via the following rule:

$$w \leftarrow w - \varepsilon\nabla_w\mathcal{L}(y, f_w(x); D),$$

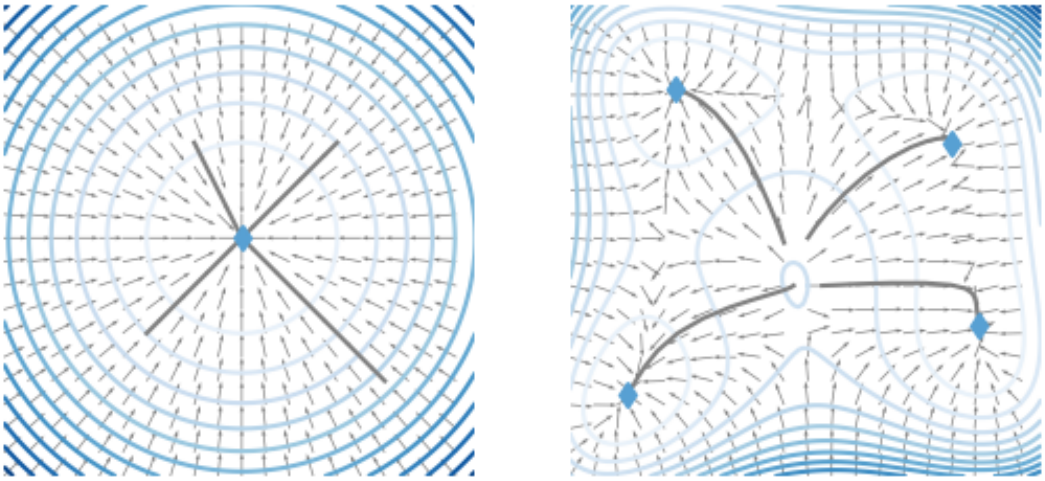


Figure 4.7. Different behaviors of gradient descent. On the left is the convex scenario, where the induced dynamics converge to the global minimum for every possible combination of initial parameters. On the right, 4 trajectories of the gradient descent dynamics for different initial conditions in the case of the Himmelbalu function. Notably, each trajectory reaches a different parameters configuration. In blue contour lines of the functions, in light gray the vector field induced by the gradient, and in solid gray the trajectories.

where ε is the *learning rate*, an optimization hyperparameter that controls the magnitude of the step in the minimizing direction. A series of considerations are now in order concerning the convergence properties of this method to the (local) minima of the loss. Indeed, the first crucial point that needs to be made is that we have no guarantee of gradient descent's converging properties for general functions. The fundamental issue is that the convergence behavior can be hindered by the presence of local minima that act as wells where the algorithm remains confined, as shown in Figure 4.7. A classical sufficient condition used to avoid this scenario and to guarantee convergence is to invoke convexity of the loss. In this case, we have the existence and uniqueness of the minimum that can therefore be reached in the limit as $\varepsilon \to 0$. Unfortunately, for real-world problems with complex losses there is no guarantee of convexity so the issue of local minima must be addressed. Nevertheless, the fundamental reason that negates the adoption of naive gradient descent for loss optimization is of a computational nature. To employ vanilla gradient descent in the training of a neural network would require staggering computational burdens for each iteration of the method, as the loss' gradients would be computed over the whole dataset D. A more sensible alternative is to approximate

Supervised Training of deep artificial neural networks

Algorithm parameters: network's weights w, $\varepsilon \in (0,1]$;
Loss $\mathcal{L}$, labels y;
foreach *epoch* **do**
  foreach *mini-batch* $\delta \in D$ **do**
    Compute forward pass of δ through the network f_w;
    Compute the loss $\mathcal{L}(y, f_w(x); \delta)$
  end foreach
  Compute stochastic estimate of the gradient $\nabla_w \mathcal{L}(y, f_w(x); \delta)$;
  Adjourn weights;
  $w \leftarrow w - \varepsilon \nabla_w \mathcal{L}(y, f_w(x); \delta)$
end foreach

the gradient of the loss with *a single sample*:

$$w \leftarrow w - \varepsilon \nabla_w \mathcal{L}(y_i, f_w(x_i); i). \tag{4.6}$$

This method evidently constitutes a stochastic approximation of the vanilla gradient descent and is therefore known as *Stochastic Gradient Descent* SGD. Notably, it is possible to prove under the reasonably mild assumption that SGD converges to vanilla gradient descent, with all the convergence considerations previously discussed. Indeed, when ε decreases at an appropriate rate and $\mathcal{L}$ is convex convergence in probability is guaranteed to the global minimum of the function [15]. In practice, SGD is employed by considering for each iteration only a subset δ of the dataset D known as *mini-batch*. This can be done efficiently with parallelization techniques and speeds up convergence to the true gradient. Algorithm 1 describes the iterative algorithm used to train a neural network with SGD. In machine learning fashion, each iteration over the dataset is known as a *epoch*. For each epoch, for each mini-batch $\delta \in D$ the outputs $f_w(x)$ of the neural network and the associated loss $\mathcal{L}(y, f_w(x); \delta)$ are computed. The loss' gradient $\nabla_w \mathcal{L}(y, f_w(x); \delta)$ is then estimated and the prescribed update performed. This procedure is repeated until either the algorithm converges to a minimum or the maximum number of epochs is reached. The following paragraph exemplifies this procedure w.r.t. a classical learning problem.

A linear example continued In paragraph 4.2.1 we demonstrated how the formal definition of neural networks given in Definition 4.2.1 can be used in its generality to describe affine functions. Here we convey how to train such functions

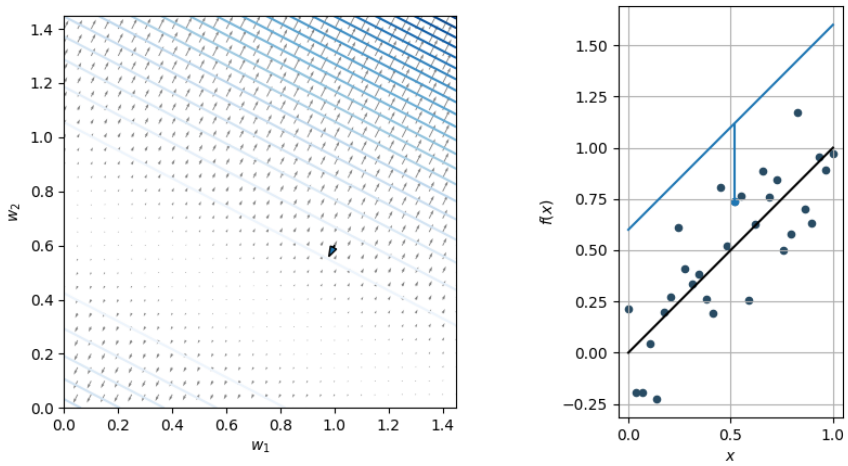


Figure 4.8. Snapshot from the training of the simple linear regressor. On the left: contour plot of the MSE loss estimated over the sampled point in dark blue shown on the right. In gray it is possible to appreciate the vector field corresponding to the estimated gradient for $w_1 = 1$, $w_2 = .6$ and $\varepsilon = .1$. On the right, in solid blue the current approximation, and in solid blue the data point used to estimate the gradient. The line segment's magnitude is the MSE. In black the minimizing fit, with $w_1 = 1$ and $w_2 = 0$.

to provide a fit to the data in the sense of simple linear regression. Recall that for a 2 input layer consisting of one bias (i.e. the affine intercept), one data input having no hidden layer and one output neuron with identity activation, the resulting approximating function takes the form:

$$f_w(x) = w_1 x + w_2.$$

We now consider a dataset $D = \{(x_1, y_1), ..., (x_n, y_n)\}$, where $x_i \in \mathbb{R}$ and assume y_i to be independent realizations drawn from omoschedastic normal random variables Y. We now consider as a loss the MSE:

$$\mathbb{E}\left[\mathcal{L}(y, f_w(x))\right] = \mathbb{E}\Big[\Big\|y - f_w(x)\Big\|_2^2\Big] = \frac{1}{n}\sum_{i=1}^{n}\left(f_w(x_i) - y_i\right)^2 = \sum_{i=1}^{n}(w_1 x_i + w_2 - y_i)^2.$$

To minimize the training loss we now consider standard SGD where the estimate of the loss gradient is computed for each data point. Figure 4.8 conveys a graphical

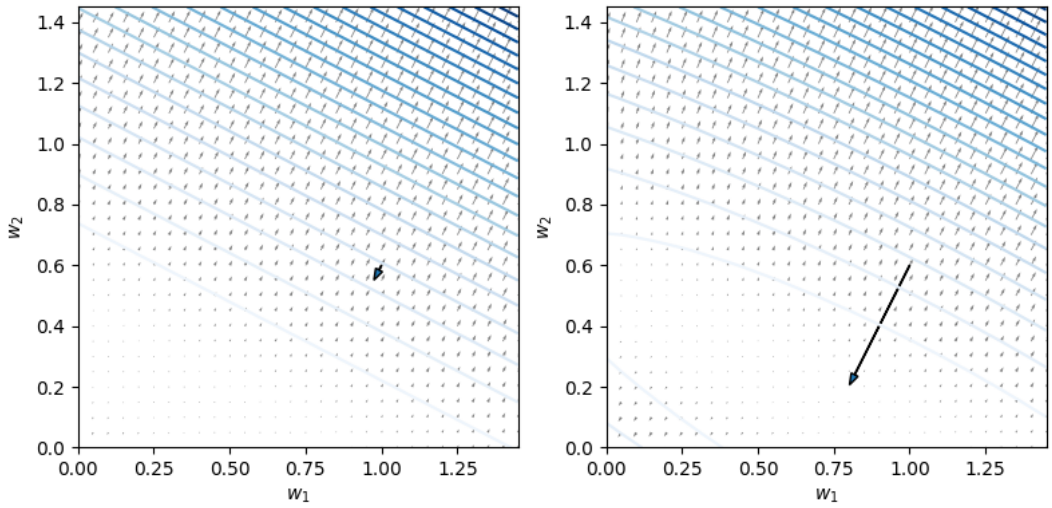


Figure 4.9. Comparison between Stochastic Gradient and vanilla gradient descent for $w_1 = 1, w_2 = 0.6$. On the left The estimate of the loss $\mathcal{L}$ obtained via single sample. On the right, the actual loss computed w.r.t. the whole dataset D. In both cases the black arrow indicates the newly reached combination of parameters

representation of this procedure.

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \varepsilon \begin{bmatrix} \frac{\partial}{\partial w_1} \mathcal{L}(y_i, f_w(x_i)) \\ \frac{\partial}{\partial w_2} \mathcal{L}(y_i, f_w(x_i)) \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \varepsilon \begin{bmatrix} \frac{\partial}{\partial w_1} (w_2 + w_1 x_i - y_i)^2 \\ \frac{\partial}{\partial w_2} (w_2 + w_1 x_i - y_i)^2 \end{bmatrix}$$
$$= \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \varepsilon \begin{bmatrix} 2x_i(w_1 x_i + w_2 - y_i) \\ 2(w_2 + w_1 x_i - y_i) \end{bmatrix}.$$

With a sensible choice of the learning rate ε, it is possible to show that training will converge to the well-known slope and intercept of the simple linear regression.

The previous example concludes the introductory section on the main features of artificial feedforward neural networks in supervised learning. In the following pages, we will focus on the method for synthetic data generation known as Adversarial Generative Networks, which is the chosen algorithm for the synthetic data generation aspect of this thesis work.

4.3 Generative Adversarial Networks

Generative modeling refers to the class of machine learning techniques that generate synthetic data by learning the probability distribution underlying training

instances. To this end, several different algorithms have been developed and employed, with varying degrees of success. From Gaussian Mixture Models to SMOTE, synthetic data generation has been a central area of interest for the machine learning community. It has therefore been natural for researchers to study possible deep learning-based strategies to address this class of problems. As the numerous generative AI tools that have sprouted in recent years prove, the use of deep learning to train generative neural networks has proven exceptionally effective, allowing for the training of models capable of passing a standardized version of the celebrated Turing's test [16]. The central position in this field is occupied by Generative Adversarial Networks (GANS). GANS are the most prevalent AI technique being used today [17] and have proven capable of mastering tasks ranging from music generation to augmenting the resolution of blurred images. In its simplest implementation, a GAN consists of two competing artificial feedforward neural networks. One is known as the *generator*, and its objective is to generate artificial data. The second network is known as the *critic*, whose objective is to discern between synthetic and real samples. The networks repeat a generate/discriminate cycle until the critic is unable to discern between artificial and training samples. We will discuss the training algorithm in detail in the following sections, after some useful definitions that are the object of the following paragraph.

4.3.1 Initial definitions

In the following pages, we will delve into the theory of Generative Adversarial Networks from the perspective of artificial feedforward neural networks, following the concepts highlighted in the previous sections. Admittedly, this choice is limiting, as it ignores architectures whose computational structure can not be represented by a directed acyclic graph, such as recurrent neural networks and long short-term machines. Nevertheless, it suffices in characterizing effectively the architecture developed in the present work.

Definition 4.3.1. We call *critic* a feedforward artificial neural network $c_\theta : \mathbb{R}^n \to \mathbb{R}$, whose trainable parameters are indexed by the pedix θ.

In this context, we assume c_θ to be a mapping between $\mathbb{R}^n$ to $\mathbb{R}$. This choice may appear limiting because images may be thought as three-dimensional tensors in $\mathbb{R}^{3,m,n}$. However, this is not the case, as we consider $\mathbb{R}^n$ to be the isomorphic space with $n = m*n*3$ via the standard flattening map. The purpose of the *critic* is to discriminate between artificial and training data. In general, the performance of the discriminator is crucial in training the generator.

Definition 4.3.2. We call *generator* a feedforward artificial neural network $g_w : \mathbb{R}^n \to \mathbb{R}^n$, whose trainable weights are indexed by the pedix w.

The generator takes as input a set of values in $\mathbb{R}^m$ and outputs the generated sample in $\mathbb{R}^n$. Usually, the input to the generator is a vector of realizations from random Gaussian noise $z \sim \mathcal{N}(0,1)$. The two networks compete in a two-player game, where each participant tries to maximize a measure of its performance known as *utility function*. Before detailing the standard training procedure employed to train GANs, a brief overview of some elementary concepts of strategic game theory is needed. In strategic form games, for each *player* i belonging to the finite set $\mathcal{V}$, a set of *actions* A_i is available. In this introduction A_i coincides with a subset of the real numbers. The set:

$$\chi = \prod_{i \in \mathcal{V}} A_i$$

is denoted *configuration space*. The vector describing the current action selected by each player is $x \in \chi$ and is called *action profile* or *configuration*. Furthermore, each player $i \in \mathcal{V}$ is equipped with a *utility function*, also known as *reward* or *payoff* or *value function*, denoted with

$$v_i : \chi \to \mathbb{R}.$$

The utility function identifies the payoff $u_i(x)$ that player i gets when each player j plays action $x_j \in A_j$. The following definition formalizes these concepts.

Definition 4.3.3. A strategic form game is a triple $G = (\mathcal{V}, \{A_i\}_{i \in \mathcal{V}}, \{u_i\}_{i \in \mathcal{V}})$, where $\mathcal{V}$ is the set of players, A_i the set of strategies available to each player and $u_i : \chi \to \mathbb{R}$ the utility function for player $i \in \mathcal{V}$.

Following standard notation

$$x_{-i} = x_{\mathcal{V} \setminus \{i\}}$$

denotes the vector obtained from action profile x by removing its $i-$th entry and,

$$v_i(x_i, x_{-i}) = v_i(x),$$

denotes the utility (or value) obtained by player i when selecting action x_i while the remainder chooses x_{-i}. These concepts are fairly general, as they encompass games having an arbitrary number of players. In games in strategic form, each player $i \in \mathcal{V}$ acts rationally in choosing the action x_i that maximizes their utility $v_i(x_i, x_{-i})$. Indeed, the player's i utility depends on the actions x_{-i} selected by the other players $j \in \mathcal{V} \setminus i$. Therefore, assuming that player i is aware of the set of actions chosen by the other players x_{-i}, and that these actions will not change, the rational behavior for player i would be to choose an element from the set:

$$\underset{x_i \in A_i}{\operatorname{argmax}}\, v_i(x_i, x_{-i}),$$

which is defined as the *best response* set. This concept can be naturally generalized by defining the set-valued best response (BR) function:

$$\mathcal{B}_i(x_{-i}) = \underset{x_i \in A_i}{\operatorname{argmax}}\, v_i(x_i, x_{-i}),$$

that formalizes the idea that players choose actions to maximize their utilities knowing the actions played by other participants in the game.

Definition 4.3.4. A Nash equilibrium (NE) for the strategic game G is an action configuration $x^* \in \chi$ such that

$$x_i^* \in \mathcal{B}_i(x_i^*), \quad \forall i \in \mathcal{V}. \tag{4.7}$$

In a Nash equilibrium, no player has *any incentive to unilaterally deviate from their current action, because the utility obtained with the current action is the best possible given the current actions selected by other players.* In general, there might be one, several, or no NE for a given game in strategic form. In the following section we will consider the problem of training a GAN within the framework provided by strategic game theory. We will discuss how a two-player game can be constructed where the actions of each player consist in providing a set of weights (i.e. a feedforward artificial neural network) and how the unique Nash equilibrium of the game can be iteratively reached. But first, we conclude the present section by considering the most common losses employed when trying to learn a distribution.

Definition 4.3.5 (Kullback-Leibler)**.** Assume X, Q random variables over the same support $\mathcal{S}$. The Kullback-Leibler (KL) divergence between the two distributions is:

$$D_{KL}\left(X \| Q\right) = \mathbb{E}_X\left[log_2\left(\frac{P_x}{Q_x}\right)\right]. \tag{4.8}$$

A direct consequence of Jensen's inequality is that, given X, Q, $D_{KL}\left(X \| Q\right) \geq 0$:

$$D_{KL}\left(X \| Q\right) = \mathbb{E}_X\left[-log_2\left(\frac{Q_x}{P_x}\right)\right] \geq -log_2\mathbb{E}_X\left[\frac{Q_x}{P_x}\right] = 0.$$

From the KL divergence, it is possible to define another distance, called Jensen-Shannon divergence.

Definition 4.3.6 (Jensen-Shannon)**.** Assume X, Q random variables on the same support $\mathcal{S}$. The Jensen-Shannon (JS) divergence is:

$$JSD(X \| Q) = \frac{1}{2}D_{KL}\left(X \middle\| \frac{X+Q}{2}\right) + \frac{1}{2}D_{KL}\left(Q \middle\| \frac{X+Q}{2}\right). \tag{4.9}$$

As we have seen, the KL divergence has a lower bound of 0, a property that the Jensen-Shannon divergence inherits directly. With these last definitions, it is possible to begin an outline of the theory behind the training of GANs.

Minibatch stochastic gradient descent training of generative adversarial networks

Algorithm parameters: generator weights w, critic weights θ, $\varepsilon \in (0,1]$;
foreach *epoch* **do**
  for k **do**
    Sample minibatch of m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from prior P_z;
    Sample minibatch from D: $\{x^{(1)}, ..., x^{(m)}\}$
    Update the discriminator by ascending the stochastic gradient:

$$\theta \leftarrow \theta + \varepsilon \nabla_\theta \frac{1}{m} \sum_{i=1}^{m} \left[log(c_\theta(x^{(i)}) + log\left(1 - c_\theta\left(g_w(z^{(i)})\right)\right)\right]$$

  end for
  Sample minibatch of m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from prior P_z. Update the generator g_w by descending the stochastic gradient:

$$w \leftarrow w - \varepsilon \nabla_w \frac{1}{m} \sum_{i=1}^{m} log\left(1 - c_\theta(g_w(z^{(i)})\right)$$

end foreach

4.3.2 Adversarial training

In the original paper from 2014 [13], the authors propose an adversarial framework based on a two-player strategic game. Under the assumption of the existence of a generating distribution P_X over the data X, it is possible to define a prior on the input noise P_z. In this original formulation, the critic c_θ is constructed to output the probability that the data input comes from X rather than $g_w(z)$. The critic is then trained to maximize the probability of assigning the correct label to both training instances and samples from g_w. The generator g_w is simultaneously trained to minimize $log(1 - c_\theta(g_w(z))$. The two networks, therefore, play a two-player minimax game having value function $v(c_\theta, g_w)$:

$$\min_w \max_\theta v(c_\theta, g_w) = \mathbb{E}_{x \sim P_X}[log(c_\theta(x)] + \mathbb{E}_{z \sim P_z}[log(1 - c_\theta(g_w(z))]. \tag{4.10}$$

Algorithm 2 details how the game is structured. For the selected number of epochs,

the critic c_θ is trained first for k iterations, as training to completion c_θ in the inner loop would result in overfitting and would be computationally prohibitive. We then optimize g_w for one step. Ideally, this approach results in c_θ being maintained close to optimality, as long as g_w changes smoothly. In the following paragraph we will see, following results presented in [13], how this game has a global optimum and how the resulting approximation learned by the networks coincides with a Nash equilibrium of the game under the assumption of *infinite capacity*. Under this assumption, subscripts are redundant so we will refer to the critic c_θ and generator g_w as c and g respectively.

Proposition 4.3.7 (**Global Optimality**). *Consider a fixed g, the optimal critic c is*

$$c_g^*(x) = \frac{P_X(x)}{P_X(x) + P_g(x)}. \tag{4.11}$$

Proof. Following Equation 4.10, the training criterion for c given g is to maximise $v(c, g)$:

$$\begin{aligned} v(c,g) &= \int_x p_X(x) log(c(x)) dx + \int_z p_z(z) log\left(1 - c(g(z))\right) dz \\ &= \int_x p_X(x) log(c(x)) + p_g(x) log\left(1 - c(x)\right) dx. \end{aligned}$$

The monotonicity of the integral implies that the maximum value of $V(c, g)$ coincides with the maximum of its argument. It is well known that $\forall (a, b) \in \mathbb{R}^2 \setminus (0,0)$, the mapping $y \to a log(y) + b log(1 - y)$ has maximum in [0,1] at $\frac{a}{a+b}$. The discriminator does not need to be defined outside the union of the two supports: $Supp(p_X) \cup Supp(p_g)$ and therefore the proof is complete. ■

Notably, the training objective for c is equivalent to maximizing the log-likelihood for estimating $P(Y = y|x)$, where Y is the Bernoulli random variable indicating whether x comes from the data P_X or the generator g. Formally:

$$\begin{aligned} C(g) &= \max_D V(g, c) \\ &= \mathbb{E}_{x \sim P_X}\left[log(c_g^*(x))\right] + \mathbb{E}_{z \sim P_z}\left[log(1 - c_g^*(z))\right] \\ &= \mathbb{E}_{x \sim P_X}\left[log(c_g^*(x))\right] + \mathbb{E}_{x \sim P_g}\left[log(1 - c_g^*(x))\right] \\ &= \mathbb{E}_{x \sim P_X}\left[log\frac{P_X(x)}{P_X(x) + P_g(x)}\right] + \mathbb{E}_{z \sim P_z}\left[log\frac{P_g(x)}{P_X(x) + P_g(x)}\right]. \end{aligned} \tag{4.12}$$

Theorem 4.3.8. *The global minimum of the training criterion 4.12 is achieved if and only if $P_g = P_X$. At that point $C(g) = -log4$.*

Proof. Checking that if $P_X = P_g$, then $C(g) = -log4$ is trivial. To see that this is the global minimum (achieved at $P_X = P_g$) for $C(g)$ we observe that:

$$\mathbb{E}_{x\sim P_X}[-log2] + \mathbb{E}_{x\sim P_g}[-log2] = -log[4].$$

By subtracting this expression from $C(g) = V(c_g^*, g)$ we obtain:

$$C(g) = -log(4) + D_{KL}\left(P_X \middle| \frac{P_X + P_g}{2}\right) + D_{KL}\left(P_g \middle| \frac{P_X + P_g}{2}\right),$$

where by KL we denote the kullback-Leibler divergence. Equivalently, recognizing the Jensen-Shannon divergence between the model's distribution and the data-generating process:

$$C(g) = -log(4) + 2JSD(P_X|P_g).$$

From the non-negativity of the Jensen-Shannon divergence, and the fact that it is zero if the two distributions are equal, we obtain the claim. ■

It is now natural to ask whether the procedure highlighted in Algorithm 2 converges to the optimum $P_g = P_X$. This is the object of the final, fundamental theorem of this theory.

Theorem 4.3.9 (**Convergence**). *Assume c and g have arbitrary representation capacity in the probability space of P_X. Assume that at each step of Algorithm 2, the critic c is allowed to reach optimum given g and that P_g is updated as to maximise:*

$$C(g) = \mathbb{E}_{x\sim P_X}\left[log(c_g^*(x))\right] + \mathbb{E}_{x\sim P_g}\left[log(1 - c_g^*(z))\right],$$

then P_g converges to P_X.

Proof. By considering $V(c, g) = U(P_g, c)$ as a function of P_g, it is possible to notice that $U(P_g, c)$ is convex in P_g. The subderivatives of a supremum of convex functions include the function's derivative at the point where the maximum is attained. This is equivalent to computing a gradient descent update for P_g at the optimal c given the corresponding g. $\sup_D U(P_g, c)$ is convex in P_g with a unique global optimum as proven in the previous Theorem 4.3.9, therefore with sufficiently small steps P_g will converge to P_X, thus concluding the proof. ■

The theoretical analysis conveyed in the previous paragraph gives reasonable guarantees concerning the convergence of the adversarial learning paradigm. Indeed, our analysis proves the existence and uniqueness of the Nash equilibrium of the game (Theorem 4.3.8) and how it can be reached (Theorem 4.3.9). Moreover, it

characterizes the optimal value to be expected once the game has reached equilibrium and it is therefore, at least in principle, an exceptionally powerful scheme. However, various issues arise in this context that make the training procedure unstable. Indeed, we have assumed from the start of the mathematical proofs that c and g can achieve arbitrary complexity, i.e. that the sequence of P_g can span the probability space where P_X is located and that c_g is capable of discerning until optimality. Unfortunately, these assumptions may be difficult to replicate in actual implementations, as the issue of complexity in representations for artificial neural networks is a non-trivial field of research. In this direction, the standard method employed to augment the representation capabilities of both critic and generator is to add neurons to the networks. Although a reasonable effort, a criterion for the choice of the architecture of either critic or generator does not exist, so the researcher is forced to deal with an iterative procedure trying to optimize the networks' architecture itself before the actual training. The second, and most important, issue arising from the schema of Algorithm 2 is the well-known [3, 13] problem of the vanishing gradients. Early in learning, when g provides a poor approximation of P_X, c can easily reject samples with high confidence because they are trivially different from the training data. This implies that $log\left(1 - c_\theta(g_w(z))\right)$ can tend to 0, thus hindering the learning process for the generator. In the original paper [13], the authors suggest to train for a few initial iterations of the generator g maximizing $log(c_\theta(g_w(z))$, which should provide stronger gradients at the beginning of the learning process. Although effective, this trick needs to be tailored to the specific problem at hand, thus increasing the overall complexity of the training scheme. A solution to this issue is to modify the underlying value function of the game. This is the approach highlighted in the following section.

4.3.3 Wesserstein GANs

In its essence, the problem addressed by generative adversarial networks is the learning of a target probability distribution P_X. To this extent, a parametric model, the generator g_w, is specified and a suitable distance metric is defined. Although not immediately obvious, this is exactly the scenario depicted by Algorithm 2 and enlightened by Theorem 4.3.8, where the equivalence between the adversarial game and the minimization of the Jensen-Shannon divergence is proven. Crucially, the Jensen-Shannon divergence is not the only metric that can be defined to discern between probability distributions. Indeed, various possible metrics can be defined, each having a different impact on the convergence of sequences of probability distributions (in our case the sequence of generators obtained by considering the weights associated with each epoch $\{g_{w_i}\}_{i\in\mathbb{N}}$). In formal terms a sequence of probability distributions $\{p_i\}_{i\in\mathbb{N}}$ on a metric space $\mathbb{S}$, it is said to converge under a metric ρ if and only if there exist a distribution $p_\infty \in \mathbb{S}$ such

that $\rho(p_n, p_\infty) \to 0$ as $n \to \infty$. The distance ρ is the fundamental discriminant in this context, as a given sequence $\{p_i\}_{i\in\mathbb{N}}$ may converge under some ρ but not under other, more strict, metric ρ'. In formal terms, we say that the topology induced that the distance ρ is *weaker* than the one induced by ρ' when the set of converging sequences under ρ is a superset of that under ρ'. It is therefore crucial to understand how strictly different common metrics act, as their implementation may hinder considerably the learning process. Indeed, in order to optimize the set of parameters w we would like to construct our generator to make the mapping $w \to g_w$ continuous. Continuity implies that if the sequence of parameters $\{w_i\}_{i\in\mathbb{N}}$ converges to a fixed point $w_\infty \in \mathbb{R}^\kappa$, then the sequence of mappings $\{g_{w_i}\}_{i\in\mathbb{N}}$ also converges to the limit distribution. Nevertheless, it is essential to remember that the notion of convergence is dependent on the topology induced by the metric, so that the weaker the induced topology, the easier will be to reach a parameter combination w_∞. In the next paragraphs, we present the theory first published in [3], as their learning algorithm constitutes the skeleton of the method employed in our research. The following sections are organized as follows: firstly I will outline the main theoretical results in favor of a learning algorithm based on the computation of the metric known as Wasserstein distance between p_{data} and g_w, instead of the procedure denoted by Algorithm 2. We will then consider two additional terms to the above-mentioned distance enforcing functional and physical constraints. Finally, the chapter will conclude with an exposition of the specific implementation developed by this thesis work.

Definition 4.3.10. We call Wasserstein or *Earth-Mover* (EM) the distance between two probability distributions p_t, p_g:

$$W(P_t, P_g) = \inf_{\gamma\in\Pi(P_t,P_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\|\right], \tag{4.13}$$

where $\Pi(P_t, P_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are P_t, P_g.

Intuitively, $\gamma(x, y)$ denotes how much mass has to be transported from x to y in order to transform one distribution into the other. The EM distance is then the cost of the optimal transport of said mass. We are now interested in the properties that the generator g_w should satisfy in order for $W(P_g, P_X)$ to be continuous and differentiable almost everywhere.

Theorem 4.3.11. *Let p_r be a fixed distribution over a compact metric set $\mathcal{X}$. Let Z be a random variable defined over $\mathcal{Z}$. Let $g_w : \mathcal{Z} \times \mathbb{R}^d \to \mathcal{X}$ be a parametric function. Let P_w be the distribution of g_w. Then:*

- *If g_w is continuous in w, so is $W(P_r, P_w)$.*

Minibatch stochastic gradient descent training of Wasserstein generative adversarial networks

Algorithm parameters: generator weights w, critic weights θ, $\varepsilon \in (0,1]$, clipping parameter c, batch size m.;
foreach *epoch* **do**
  for k **do**
    Sample minibatch of m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from prior P_z;
    Sample minibatch from D: $\{x^{(1)}, ..., x^{(m)}\}$
    Update the discriminator c_θ

$$\theta \leftarrow \theta + \alpha \nabla_\theta \frac{1}{m} \sum_{i=1}^{m} \left[c_\theta(x^{(i)}) - c_\theta(g_w(z^{(i)})) \right]$$

    then clip the updates:

$$\theta \leftarrow clip(\theta, -c, c)$$

  end for
  Sample minibatch of m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from prior P_z. Update the generator g_w:

$$w \leftarrow w - \alpha \nabla_w \frac{1}{m} \sum_{i=1}^{m} c_\theta(g_w(z^{(i)}))$$

end foreach

- *If g is locally Lipschitz and continuous in w, then $W(P_r, P_w)$ is continuous and differentiable almost everywhere.*

- *The first and second statements are false for the Jensen-Shennon and KL divergences.*

Proof. We consider two parameter vectors $w, w' \in \mathbb{R}^d$. To prove the theorem we will find a bound on $W(P_w, P_{w'})$. The main idea is to use the coupling γ. Indeed, the distribution of the random variable $(g_w(z), g_{w'}(z))$, which has $\gamma \in \Pi(P_w, P_{w'})$.

From the definition:

$$\begin{aligned} W(P_w, P_{x'}) &\leq \int_{\mathcal{X}\times\mathcal{X}} \|x-y\| d\gamma \\ &= \mathbb{E}_{(x,y)\sim\gamma}\left[\|x-y\|\right] \\ &= \mathbb{E}_z\left[\|g_w(z) - g_{w'}(z)\|\right]. \end{aligned}$$

If g is continuous in w, then the convergence of $w \to w'$ in the $\mathbb{R}^d$ euclidean norm implies the pointwise convergence $\|g_w - g_{w'}\| \to 0$. We have hypothesized $\mathcal{X}$ to be compact, so that the distance of any two elements in it must be bounded by some constant M: $\|g_w - g_{w'}\| \leq M$ for all w and z uniformly. By the bounded convergence theorem we have therefore:

$$W(P_w, P_{w'}) \leq \mathbb{E}_z\left[\|g_w(z) - g_{w'}(z)\|\right] \to_{w\to w'} 0,$$

and by the triangular inequality:

$$|W(P_r, P_w) - W(P_r, P_{w'})| \leq W(P_w, P_{w'}) \to 0.$$

Thus the continuity of $W(P_r, P_w)$ is proven. Now we consider the differentiability. We assume g to be Lipschitz, so that for a pair (w, z) we have a constant $L(w, z)$ and an open set $U(w, z)$ so that $\forall(w', z') \in U$ it holds:

$$\|g_w(z) - g'_w(z)\| \leq L(w, z)(\|w - w'\| + |z - z'\|).$$

Requiring $z' = z$ we have in expectation:

$$\mathbb{E}_z\left[\|g_w(z) - g_{w'}(z)\|\right] \leq L(z, w)\mathbb{E}_z\left[L(z, w)\right],$$

$\forall(w', z') \in U$. At this point, we can consider the restriction $U_w = \{w'|(w', z) \in U\}$, and as U was supposed ope, so has to be U_w. The following bound comes then naturally:

$$|W(P_r, P_w) - W(P_r, P_{w'})| \leq W(P_w, P_{w'}) \leq L(w)\|w - w'\|,$$

for all $w' \in U_w$. This implies that $W(P_r, P_w)$ is indeed locally Lipschitz, and therefore locally continuous and Radamacher's theorem implies differentiability almost everywhere. To prove the final statement it suffices to consider a counterexample provided in [3]. ■

In the following corollary we see that optimization-based learning by minimizing the EM distance is sensible.

Corollary 4.3.12. *Assume g_w to be an artificial neural network parametrized by w and P_z a prior over z such that $\mathbb{E}_{z\sim P_z}[z] < \infty$. Then g_w is continuous in w and $W(P_r, P_w)$ is differentiable almost everywhere.*

The proof can be found in [3]. This Corollary proves that the EM distance is naturally amenable to gradient-based optimization methods, as it achieves differentiability almost everywhere in contrast with the Jensen-Shannon and KL metrics. The final piece of the picture is a characterization of the relative strengths of the topologies induced by each distance. Ideally, we would like to show that the Wasserstein distance induces the strongest topology amongst the considered metrics because this would in turn imply that the convergence in EM is stronger than each of the others. This is the object of the following theorem.

Theorem 4.3.13. *Let P be a probability distribution over a compact space $\mathcal{X}$ and $\{P_n\}_{n\in\mathcal{N}}$ be a sequence of distributions on $\mathcal{X}$. Then the following holds:*

$$D_{KL}(P_n\|P) \to 0 \implies JSD(P, P_n) \to 0 \implies W(P, P_n) \to 0. \tag{4.14}$$

Furthermore, $P_n \to P$ in distribution.

This exactly confirms our hopes, as Theorem 4.3.13 from [3] exactly proves that the EM distance induces the strongest topology out of all the considered metrics. We shall now discuss how to approximate this metric in practice to effectively train the generator and discriminator.
Although the theoretical properties of the Wasserstein metric prove that it is the best available choice between JS and KL, the definition given in [3] is highly intractable. On the other hand, a celebrated result known as Kantorovich-Rubenstain duality [8] states that

$$W(P_r, P_w) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x\sim P_X}\left[f(x)\right] - \mathbb{E}_{x\sim P_w}\left[f(x)\right],$$

where the supremum is over all the k-Lipschitz functions $f : \mathcal{X} \to \mathbb{R}$. Therefore, if we have a parametrized family of functions, such as c_θ and g_w, that are all Lipschitz for some K, we would be solving the problem

$$\max_{\theta\in\Theta} \mathbb{E}_{x\sim P_X}\left[c_\theta(x)\right] - \mathbb{E}_{z\sim P_z}\left[c_\theta(g_w(z)\right],$$

and if the supremum is attained for some combination of parameters θ this process would yield a calculation of the EM distance modulo a multiplicative constant K. Furthermore, we could differentiate $W(P_r, P_g)$ with respect to the generator's parameters w and therefore apply gradient descent methods to train the generator itself. The following theorem proves this intuition.

Theorem 4.3.14. *Let P_r be a probability distribution over $\mathcal{X}$. Let P_w be the probability distribution of g_w. Then there is a solution $f : \mathcal{X} \to \mathbb{R}$ to the problem:*

$$\max_{\|f|\leq 1} \mathbb{E}_{x\sim P_r}\left[f(x)\right] - \mathbb{E}_{x\sim P_w}\left[f(g_w(x))\right],$$

and we have:

$$\nabla_w W(P_r, P_w) = -\mathbb{E}_{z\sim P_z}\left[\nabla_w f(g_w(z))\right].$$

Proof. We begin by defining:

$$\begin{aligned} V(\hat{f}, w) &= \mathbb{E}_{x\sim P_r}\left[\hat{f}(x)\right] - \mathbb{E}_{x\sim P_w}\left[\hat{f}(x)\right] \\ &= \mathbb{E}_{x\sim P_r}\left[\hat{f}(x)\right] - \mathbb{E}_{z\sim p(z)}\left[\hat{f}(g_w(x))\right], \end{aligned}$$

where $\hat{f}$ lies in $\mathcal{F} = \left\{\hat{f} : \mathcal{X} \to \mathbb{R}, \hat{f} \in C_b(\mathcal{X}), \|\hat{f}\|_L \leq !\right\}$ and $w \in \mathbb{R}^d$. The compactness of $\mathcal{X}$ guarantees, via the Kantorovich-Rubinstein duality that there is an $f \in \mathcal{F}$ that attains the value:

$$W(P_r, P_w) = \sup_{\hat{f}\in\mathcal{F}} V(\hat{f}, w) = V(f, w).$$

If we now define $X^*(w) = \{f \in \mathcal{F} : V(f, w) = W(P_r, P_w)\}$, we know that $X^*(w)$ is non-empty. Using an envelope theorem:

$$\nabla_w(P_r, P_w) = \nabla_w V(f, w),$$

for any $f \in X^*(w)$ when both terms are well defined. Now let $f \in X^*(w)$, which exists since $X^*(w)$ is non-empty for all w, then:

$$\begin{aligned} \nabla_w W(P_r, P_w) &= \nabla_w V(f, w) \\ &= \nabla_w \left[\mathbb{E}_{x\sim P_r}\left[f(x)\right] - \mathbb{E}_{z\sim p(z)}\left[f(g_w(z))\right]\right] \\ &= -\nabla_w \mathbb{E}_{z\sim p(z)}\left[f(g(_w(z))\right]. \end{aligned}$$

With fine technical arguments based on a clever application of Fubini's theorem (that we do not provide in this context), it is possible to conclude the proof by showing that the operator ∇ commutes with the expected value:

$$-\nabla_w \mathbb{E}_{z\sim p(z)}\left[f(g(_w(z))\right] = -\mathbb{E}_{z\sim p(z)}\left[\nabla_w f(g(_w(z))\right].$$

Details can be found in [3]. ■

Theorem 4.3.14 is the crown achievement reported in [3]. It proves that gans can be trained using the EM distance and that in so doing we can expect to obtain better results than the ones guaranteed by training via Jensen-Shannon minimization. The main difference between the vanilla implementation of a generative adversarial network's training algorithm and its Wasserstein counterpart can be appreciated under the game theoretical lens that we outlined in the previous sections. In the vanilla game the critic $c_\theta : \mathbb{R}^n \to [0,1]$ outputted a probability that served as a measure of its confidence in the nature of the input samples. The generator g_w on its part was trained to produce convincing samples to fool the critic. The final form of the game's value function was therefore:

$$\min_g \max_c V(c_\theta, g_w) = \mathbb{E}_{x\sim P_X(x)}[log(c_\theta(x)] + \mathbb{E}_{z\sim P_z}[log(1 - c_\theta(g_w(z))].$$

Minibatch stochastic gradient descent training of Wasserstein generative adversarial networks with gradient penalty

Algorithm parameters: generator weights w, critic weights θ, $\varepsilon \in (0,1]$, regularization parameter λ, batch size m.;
foreach *epoch* **do**
  for k **do**
    Sample minibatch of m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from prior p_σ;
    Sample minibatch from D: $\{x^{(1)}, ..., x^{(m)}\}$
    Sample $\hat{x}$ from $x^{(i)}$: $\hat{x} = \epsilon x + (1-\epsilon)\ g_w(z^{(i)})$
    Update the discriminator c_θ

$$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} \left[c_\theta(x^{(i)}) - c_\theta(g_w(z^{(i)})) \right] + \lambda \mathbb{E}_{\hat{x} \sim p_{\hat{x}}} \left[\left(\|\nabla_{\hat{x}} c_\theta(\hat{x})\|_2 - 1 \right)^2 \right]$$

  end for
  Sample minibatch of m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from prior P_z. Update the generator g_w:

$$w \leftarrow w - \alpha \nabla_w \frac{1}{m} \sum_{i=1}^{m} c_\theta(g_w(z^{(i)}))$$

end foreach

In a Wasserstein gan the fundamental change happens in the value function for the game:

$$textmin_g \max_c V(c_\theta, g_w) = \min_w \max_{\|c_\theta\| \leq 1} \mathbb{E}_{x \sim P_X} \left[c_\theta(x) \right] - \mathbb{E}_{x \sim P_z} \left[c_\theta(g_w(x)) \right].$$

In this context, we are not bounding the critic to a mapping between $\mathbb{R}^n$ and [0,1]. On the contrary, the critic is allowed to take arbitrary values (up to Lipschitz compliance), so that the higher the output's module, the more certain the critic is of the provenience of the input data. In practice, the fundamental constraint to be enforced is the belonging of the critic to the class of Lipschitz functions. In the original per on Wesserstein GANs the authors suggested the method of gradient clipping to avoid the violation of this assumption. Algorithm 3 details the training process. The idea of clipping the gradients, although of simple implementation, can be detrimental as the choice of the clipping parameter affects strongly the convergence of the training procedure. With permissive values, the critic may initially

converge faster, without reaching convergence. On the other hand, less permissive choices may hinder the convergence, requiring unacceptable amounts of epochs. A solution to this problem was found by [14]. As they note, the Wasserstein GAN optimization process is difficult because of the interactions between the weight constraint and the cost function, which result in either vanishing or exploding gradients without a sensible choice of the clipping parameter. Their proposal to enforce Lipschitz constraints is based on the fact that a differentiable function is 1-Lipschitz if and only if it has gradients with norm 1 almost everywhere. They directly constrain the gradient norm of the critic's output with respect to its input. The final loss for the critic becomes:

$$\max_{\theta\in\Theta} \mathbb{E}_{x\sim P_X}\left[c_\theta(x)\right] - \mathbb{E}_{z\sim P_z}[c_\theta\left(g_w(z)\right)] + \lambda\mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\left[\left(\|\nabla_{\hat{x}}c_\theta(\hat{x})\|_2 - 1\right)^2\right], \tag{4.15}$$

where λ controls the strength of the regularizing action and $\hat{x} = \epsilon x + (1-\epsilon)g_w(\sigma^{(i)})$ with $\epsilon \in (0,1)$. This method is now known as *gradient penalty* and has proven exceptionally effective in stabilizing the learning dynamics of Wasserstein gans.
Wasserstein GANs trained via gradient penalty offer a reliable framework for synthetic data generation. They can be applied to a variety of different problems, with satisfactory results. However, their flexibility is limited when considering data that has to comply with strict constraints. In scenarios of this kind, the generator usually learns an unsatisfactory approximation of the data distribution, producing outputs that fail to comply with the limits imposed by the system. An example of this behavior can be appreciated in the context of physically informed neural networks, that are used by researchers to emulate physical phenomena. With a loss in the form of Equation 4.15, the generator fails to produce data compliant with the physical phenomenon of interest. To address this final limitation, the general approach pioneered by the research community has been to add a regularizing term to the global value function of the game:

$$\begin{aligned} v(c,g) = \max_{\theta\in\Theta} \mathbb{E}_{x\sim P_X}\left[c_\theta(x)\right] - \mathbb{E}_{z\sim P_z}[c_\theta\left(g_w(z)\right)] + \lambda\mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\left[\left(\|\nabla_{\hat{x}}c_\theta(\hat{x})\|_2 - 1\right)^2\right] - \\ \mu\mathbb{E}_{z\sim p(z)}\left[H(g_w(z))\right]. \end{aligned} \tag{4.16}$$

This is the general form of the adversarial game employed in this thesis work. The novel term $\mathbb{E}_{z\sim p(z)}\left[H(g_w(z))\right]$ behaves like a regularizing term of strength mediated by the parameter μ, as the generator is now required to maximize the linear combination:

$$\mathbb{E}_{z\sim P_z}\left[\nabla_w c_\theta(g_w(z))\right] + \mu\mathbb{E}_{z\sim P_z}\left[\nabla_w H(g_w(z))\right].$$

In this context the operator $H : \mathbb{R}^n \to \mathbb{R}$ is supposed differentiable almost everywhere and represents the hypersurface whose gradient coincides with the constraints to be enforced.

Minibatch stochastic gradient descent training of conditional Wasserstein GANs with gradient penalty and physical constraints

Algorithm parameters: generator weights w, critic weights θ, $\varepsilon \in (0,1]$, regularization parameter λ, batch size m.;
foreach *epoch* **do**
  for k **do**
    Sample minibatch of m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from prior p_σ;
    Sample minibatch from D: $\{x^{(1)}, ..., x^{(m)}\}$
    Sample $\hat{x}$ from $x^{(i)}$: $\hat{x} = \epsilon x + (1-\epsilon)\ g_w(z^{(i)}|q^{(i)})$
    Update the discriminator c_θ

$$\theta \leftarrow \theta - \nabla_\theta \frac{1}{m} \sum_{i=1}^{m} \left[c_\theta(x^{(i)}|q^{(i)}) - c_\theta(g_w(z^{(i)}|q^{(i)})) \right] + ... + \lambda \mathbb{E}_{\hat{x} \sim p_{\hat{x}}} \left[(\|\nabla_{\hat{x}} c_\theta(\hat{x})\|_2 - 1)^2 \right]$$

  end for
  Sample minibatch of m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from prior P_z. Update the generator g_w:

$$w \leftarrow w - \epsilon \left[\nabla_w \frac{1}{m} \sum_{i=1}^{m} \left(c_\theta(g_w(z^{(i)}|q^{(i)})) + \mu H(g_w(z^{(i)}|q^{(i)})) \right) \right]$$

end foreach

Equation 4.16 concludes the introductory section on Generative Adversarial Networks and their training. We will now discuss how these concepts can be fruitfully employed to generate synthetic initial conditions for the benchmark SUMO scenario considered in this work.

4.3.4 Conditional Wasserstein GANs

Until this point, we have discussed the problem of learning a probability distribution in the general case where a target distribution Y is approximated by the sequential refining of the generator g_w. In the applications, however, we would

like to specify a class and draw samples that belong to the class itself, rather than obtaining an arbitrary realization. Indeed, our current exposition describes $g_w(z) \to \mathbb{R}^n$, where z is drawn from the normal distribution having mean 0 and variance 1. By computing the image of z though g_w we obtain a realization (supposing that convergence has been reached) from Y. Let us suppose now that the support of Y is the space of all the possible grayscale images having size 28×28 pixels representing handwritten digits. In our current implementation, when we compute $g_w(z)$ we have no control over which number (i.e. image depicting the number) will be generated. This issue is of importance because we would like to be able to specify the class to which a given sample should belong. Fortunately, solving this issue is straightforward, as it is sufficient to consider, rather than simply z, the concatenation of z and a one hot encoded label q corresponding to the class of interest. Formally, the global value function of the game changes slightly:

$$v(c,g) = \max_{\theta \in \Theta} \mathbb{E}_{x \sim P_X} \left[c_\theta(x|q) \right] - \mathbb{E}_{z \sim P_z} [c_\theta \left(g_w(z|q) \right)] + \lambda \mathbb{E}_{\hat{x} \sim P_{\hat{x}}} \left[\left(\|\nabla_{\hat{x}} c_\theta(\hat{x}|q)\|_2 - 1 \right)^2 \right] - \mu \mathbb{E}_{z \sim p(z)} \left[H(g_w(z|q)) \right]. \tag{4.17}$$

In Equation (4.17) the notation $x|q$ is to be understood as the "concatenation of white noise z and the one hot label for class i q". Notably, the insertion of the term q does not affect the general training algorithms developed by the theory as can be seen in Algorithm 5, for it can be considered as a projection operator that forces the mapping g_w to some kind of specific submanifold belonging to the support of Y. With this final remark, the theory of Wasserstein GANs is now complete. We will address in the following Chapter the implementation that has been devised in the context of this thesis work.

Chapter 5

Experimental Results

This Chapter is subdivided into two sections. In Section 5.1 the implementation pipeline is explained in detail. We discuss the generation of both pedestrian and vehicular demand and the associated parameters employed in the subsequent simulations. Furthermore, we define the data acquisition scheme employed in this work and the dataset generation process. Section 5.2 defines the employed architecture and compares its effectiveness in generating risk scenarios with the standard baseline defined by the demand generation procedure.

5.1 Setup

The main objective of the present work is to present an effective framework for the generation of configuration files (i.e. initial values) that, when fed to SUMO, will cause (on average) pre-specified collision events. To this extent, an ML pipeline has been developed to train a Generative Adversarial Network to generate such files. Crucially, the fundamental building block of any learning-based algorithm is data. The present Section deals with the assumptions and techniques devised to build the dataset used in the training of the proposed GANs. Simulations take place in the road network described in Chapter 2, with 4 lanes leading to a central regulated intersection. Pedestrians can walk in any direction alongside each lane and can cross the intersection at designated crossings. As previously mentioned, standard SUMO simulations do not allow for collision events between agents. Indeed, the simulation itself has been developed with the explicit aim of avoiding such events to simplify its internal dynamics. However, SUMO allows for a notable amount of freedom when specifying the parameters of the models that govern each agent's dynamics, so that it is actually possible to generate a variety of risk scenarios, including collisions and emergency braking. To this extent, in the present work we have employed the default pedestrian (striping) and Krauss's [20]

Vehicle parameters					
acc=7.0	dec=5.0	mSpeed=18.0	mGap=2.5	σ=0.5	JmIgProb=0.6

Table 5.1. Parameters specified for each vehicle. Σ refers to the standard deviation of the normal distribution from which each vehicle's velocity is drawn.

car-following models. Table 5.1 reports the chosen parameters for the latter. As it is possible to appreciate, we are considering realistic vehicles, with acceleration and deceleration compatible with real-world counterparts. Indeed, a maximum acceleration has been set equal to $7m/s^2$, with a corresponding braking deceleration of $4m/s^2$. It is important to remark that $7m/s^2$ is the maximal acceleration that a vehicle can reach, in practice the limitations imposed by the car-following model imply that vehicles rarely reach the maximal acceleration. Similar considerations can be repeated for the maximal deceleration. In the proposed road network accidents take place at the intersection, where vehicles may collide with each other or with pedestrians. As we have discussed in Chapter 2, the behavior of vehicles at intersections can be modeled by specifying the likelihood for a vehicle to ignore the right of way implied by the intersection itself. Whenever such an event happens, the incoming vehicle collides either with a pedestrian or with another vehicle. The gravity of said event can be measured using the classification system detailed in Chapter 3. On the topic of demand generation, we employ a custom script that balances the number of generated vehicles on every incoming lane, sampling from a Poisson distribution and feeding it to the duarouter method. A similar approach has been used to generate pedestrian demand, using the randomTrips utility provided by SUMO that generates a random number of pedestrians for each simulation.

We can now outline the pipeline developed to generate the dataset. The process starts with the generation of pedestrian and vehicular demand. The simulation is then run and the logfiles (containing the logs of the events that happened during the simulation itself) are extracted. A script is then called that checks whether or not the simulation contains collision events and, if present, saves the timesteps at which they happened and stores an XML file containing the positions, angle, and velocity of each agent. By computing the ΔV for each vehicular collision event and the incoming vehicle's velocity for pedestrian collisions, we can then assign a label to each configuration. Crucially, this process is repeated for *every event within the simulation*, so that from a single simulation, multiple XML snapshots can be recovered. The process is repeated a prescribed number of times and the resulting data is stored. Once the process has ended, the user is then presented with a labels folder containing json files and a folder of XML files representing the snapshots previously saved. Notably, each snapshot consists of a different number of vehicles

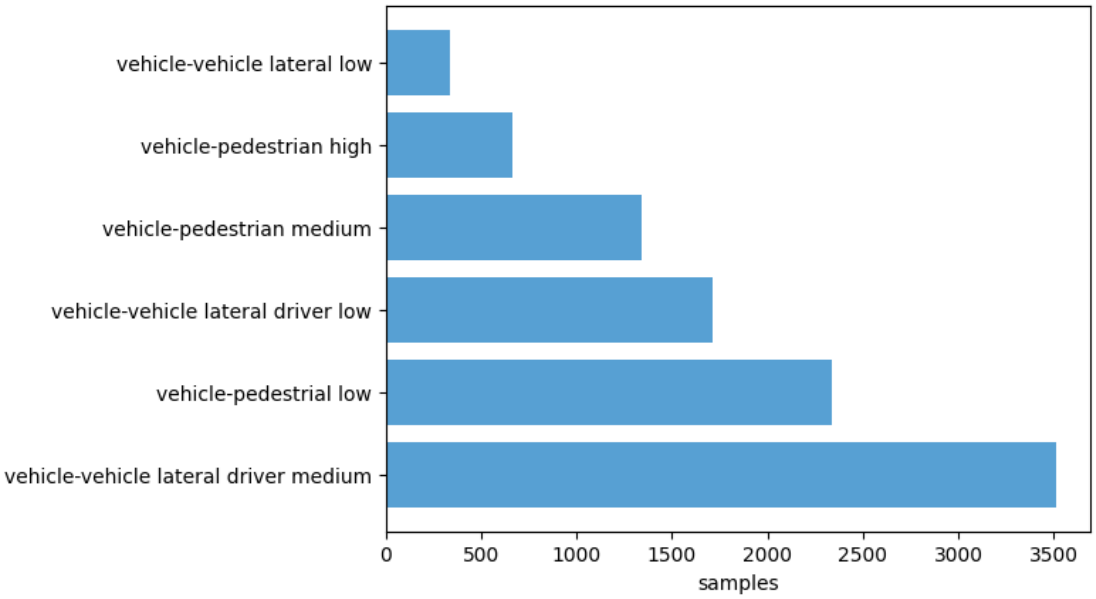


Figure 5.1. Class distribution for the final 9902 samples dataset. Crucially, almost half of the data leads to a collision of medium gravity between vehicles on the driver's side.

and pedestrians, so the problem of varying data length arises. We propose to specify a common threshold for the number of vehicles and pedestrians that need to be present for a snapshot to be saved. For example, given an XML file containing the positions, angles, and velocities of each agent, we convert it to a tensor only if the number of pedestrians (and vehicles) is greater or equal to a given value. Moreover, if the number of each kind of agent exceeds the threshold t, we store only the t closest to the intersection. This is an effective strategy, for a locality assumption can be upheld. Indeed, we are saving the state of the simulation at Δt seconds before the event takes place, so it is reasonable to assume that only those vehicles close to the intersection will influence directly the dynamics of the system. It is important to remark that in this seminal work, we are limiting the scope of the network to the generation of collision events. This limitation is justified by the fact that the generation of emergency braking scenarios in SUMO is straightforward as it is only required to specify high velocities and short intervehicular distances. At simulation time, the car following the model will invariably reduce the speed of the chasing vehicle, thus giving rise to emergency braking events.

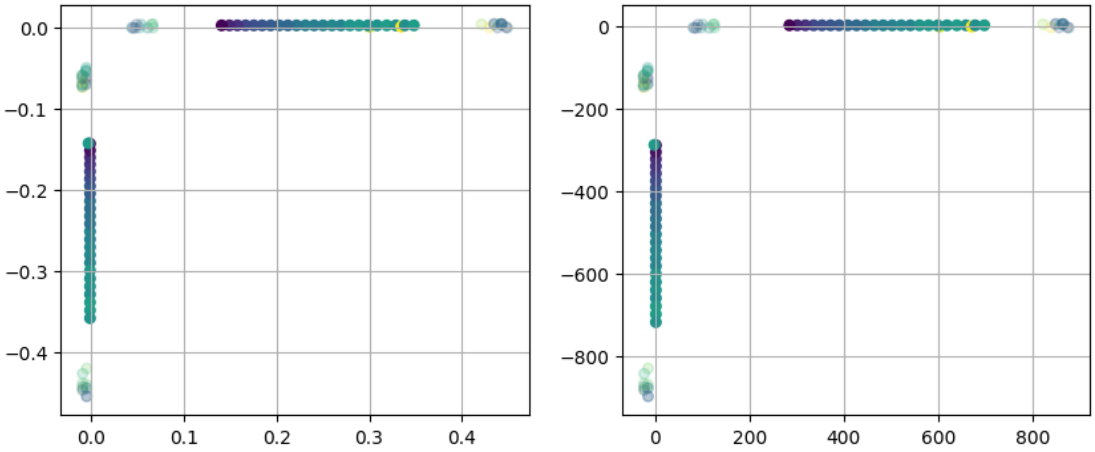


Figure 5.2. Plot of a sampled tensor from the dataset. Each point represents either a vehicle (solid hue), or a pedestrian (lighter hue), and it is colored based on their velocity. *On the left*: the rescaled data. *On the right*: original data.

5.1.1 Dataset

To generate the dataset we made use of an SSH server equipped with an Intel Xeon Processor (Skylake). The resulting data consists of 10556 individual configurations leading each to one of the six possible events. The subsequent filtering, with a fixed number of 50 vehicles and 50 pedestrians, leads to a final dataset comprising 9902 labeled tensors. The bar plot in Figure 5.2 conveys the relative class distribution. Crucially, the dataset is highly unbalanced. This is to be expected: the combination of each vehicle's parameters, the topology of the road network, and the right-of-way regulations inevitably favor specific events over all possible ones. This phenomenon is akin to what happens in real-world road networks, where a clustering of accidents sharing common characteristics is commonly observed. From a learning perspective, however, this phenomenon constitutes a notable hindrance. Indeed, research shows [5] that extreme imbalances between classes, such as the ones present in our study, can disrupt significantly the approximation learned by the generator. A standard strategy that can be helpful in mitigating those effects consists of a biased sampling of the training data, which will be discussed in the Section reserved for our training methods.

5.1.2 Preprocessing

As previously noted (see Chapter 2), SUMO employs various coordinate systems. In our implementation, we have saved the positions, velocities and angles of each vehicle and pedestrian. In principle, the adimensional positions for each agent can

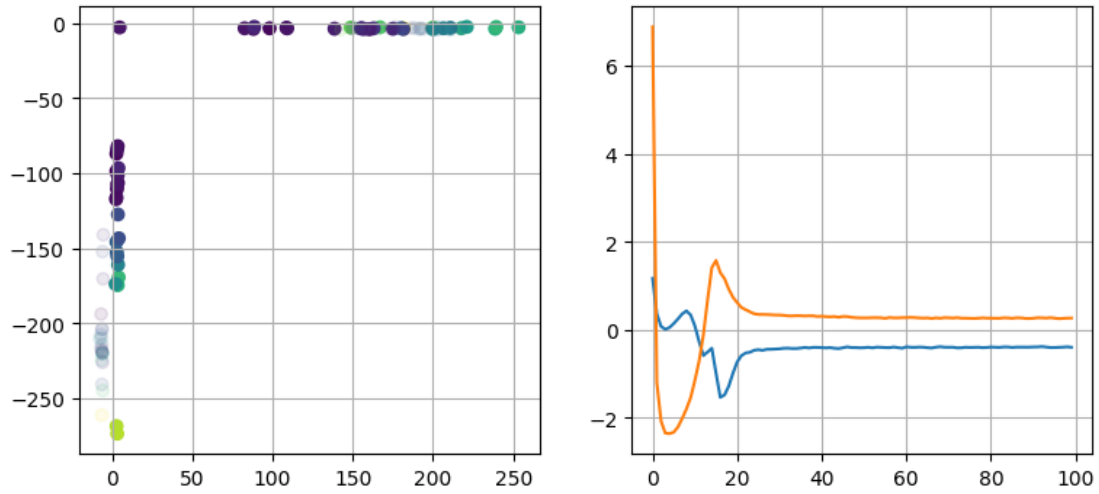


Figure 5.3. *On the left*: generated data for the "vehicle vehicle lateral driver medium" class. *On the right*: averaged losses by mini batch for each epoch of the critic (orange) and generator (blue).

vary between -1000 to 1000, whereas the velocities and angles all span different orders of magnitude so normalization is necessary. To this end we performed a minimax rescaling over the whole dataset, bounding each feature between 1 and -1. This preprocessing technique has proven crucial in aiding the GAN to progress in its learning dynamics. Furthermore, the coordinate system employed by SUMO for pedestrians and vehicles shifts the x and y coordinates with respect to the center of the pathway/lane. Although in principle of limited impact, this small translation has proven significant in developing the regularization therm $H(g_w)$ specified in the previous Chapter. In this direction, with the idea of simplifying H, we have translated the coordinate system so that x and y represent the distance from the axis passing through the lanes (and not each lane's central axis).

5.1.3 GAN

The development of the Generative Adversarial Network at the core of the presented work has progressed incrementally. We first began by considering a reduced version of the dataset comprised of just over 2000 instances to gauge the required complexity. The initial architecture of the network consisted of two vanilla feedforward deep artificial neural networks, one for the generator and one for the critic. Nevertheless, by training the network on this arbitrary subset of the generated data we were not able to produce satisfactory results. Indeed, the critic would

inevitably overfit the training data leading to diverging losses. As our understanding of the complexities of the learning problem increased, we consequently shifted from the vanilla generator to a more complex architecture, that is depicted in Figure A.1. This architecture has been trained on the whole dataset and the results described in the following Section refer to this topology.

5.1.4 Training

Arguably, the most delicate aspect of this work has been devising a stable training procedure. In this direction, the first step has been the implementation of a biased sampling method to try and mitigate the dataset's imbalance. By assigning to each of the 9902 instances the relative frequency of its class in the dataset and sampling with probability proportional to the inverse of such weight we can sample mini-batches from the dataset that have uniform class distribution. In so doing we can mitigate the effects of the class imbalance observed in the data and stabilize the networks' training. To optimize each network's weights we have employed Adam optimizer with learning rate $lr = 0.00005$ prescribed in [3, 12, 13] and a coefficient $\lambda = 10$ for the gradient penalty term, coupled with a scheduler to reduce the learning rate (Lr scheduler) by 10% each epoch. For each iteration of the generator's training, we update the critic for 20, following the suggestions found in [13]. Initial tests using this strategy have shown promising results, demonstrating how the network could at least partially infer the structure underlying the data. It is paramount to recall the finality of the synthetic data: that it is to be transformed into a set of configuration files for a simulator. In particular, the generator needs to output a set of velocities and positions that must comply to physical constraints: they must belong to a set of acceptable values. This strict requirement has been the central issue in the development and training of the presented architecture and the fundamental motivation leading to the adoption of the regularizing

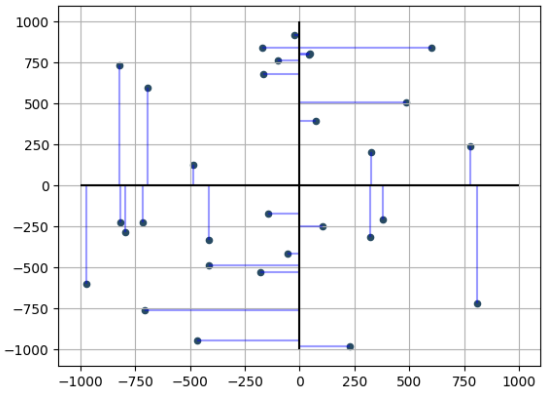


Figure 5.4. Example of the regularization term's action on 30 agents: the term H computes the cumulative squared distances (in light blue) of each agent from the center of its lane.

term $\mathbb{E}_{z\sim P_z}[H(g_w(z))]$ originated in the physics literature. In the presented implementation we defined H as the cumulative squared distance of each point representing an (agent) from the center of its closest lane (see Figure 5.1.4), weighted by a constant $\alpha = 0.01$. In so doing we effectively have managed to train the network to comply with the constraints enforced by the simulator. Figure 5.1.4 conveys the convergence of the training procedure and the resulting generated scenario. It is possible to notice how the generator's loss falls below the critic's, demonstrating how the learning procedure has produced a generator capable of consistently fooling the critic itself.

5.2 Results

As we have previously discussed, the generator's output consists of a $2D$ matrix whose features represent the positions and velocities of each agent within the road network. To effectively perform a simulation based on the generated data we convert the output to a series of 3 configuration files, including only the agents that comply with SUMO's constraints. Notably, the training procedure devised in the context of the present work has proven effective in guaranteeing that, normally, all vehicles and pedestrians are correctly placed. Once the files have been written, the simulation is run and the log file is stored. To validate the performance of our network we perform a statistical analysis on the simulation's output on two aggregation levels: one general and the other specific. For the general case, we ask the network to generate configurations that lead to events belonging to the same macro collision class: either vehicle-vehicle or vehicle-pedestrian. The simulation's log files are then stored and the empirical likelihood of the vehicle-vehicle (or vehicle-pedestrian) collision event is estimated within a 95% Wilson confidence interval. The Wilson test is a binomial proportion confidence interval, i.e. a confidence interval for the probability of success calculated from the outcome of a series of success–failure experiments (Bernoulli trials). In this context, the number of Bernoulli trials coincides with the total number of collision events recorded during the unfolding of the simulations and the success probability with the fraction of events coinciding with the class specified. The choice of Wilson's test over other possible alternatives, such as Clopper-Pearson or the standard Wald interval is motivated by its intrinsic robustness. The idea behind this first evaluation task is to assess whether the network is capable of consistently generating initial conditions leading to a macro area of events. For the specific evaluation, on the other hand, we first select a specific class of event that we would like the simulator to generate (a low-gravity collision between a pedestrian and a vehicle for example). We then run a predetermined number of simulations, generating for each configuration file using the network. For each simulation, we store the log files and classify

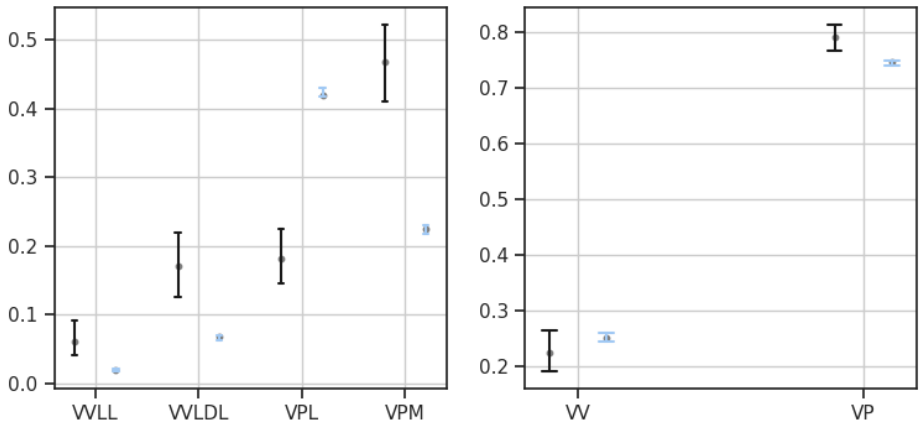


Figure 5.5. *On the left*: plot of the 95% confidence interval for selected classes. GAN's values are in black, whereas control is in blue. Probabilities are the gray dots. *On the right*: plot of probabilities (gray dots) and confidence intervals (95%) for the general test. It is possible to appreciate the partial success obtained by the generator.

the happened events. Once the total number of iterations has been reached, we compute the empirical fraction of events of the requested type over all the events generated and compare it with the baseline from the dataset. To ensure statistical significance the comparison is carried out by computing 95% Wilson confidence intervals for each of the classes present in the dataset. Before presenting the results of the evaluation procedure it is paramount to recount that for every generated initial configuration, a simulation is run in order to classify each happened event. Therefore, to avoid impractical computation times, the number of iterations for each evaluation has been defined accordingly.

5.2.1 Discussion

We performed a statistical evaluation of our method comparing its results with the events of another dataset generated using the same parameters of the one used to train the network, with the only exception of having a number of vehicles and agents fixed at 50. The rationale behind this choice is to provide an effectively unbiased benchmark that estimates the specific collision events distribution resulting from said parameters. This novel dataset is comprised of 2740 different instances spanning the same 6 collision classes of its fellow used at training time. The results of the first test we proposed are depicted on the right side of Figure 5.5. For this case, we have asked the network to generate either a collision event between

	Random	GAN
Vehicle-Vehicle	0.712	**0.83**
Vehicle-Pedestrian	2.028	**3.09**

Table 5.2. Comparison of control and GAN generated collision events grouped by macro classes. The GAN proves to be significantly more effective in generating pedestrian collision scenarios.

vehicles or vehicle-pedestrian for 100 times each. In the case of vehicle-vehicle collisions, we reported a total of 365 generated events, with a 22.5% incidence of inter-vehicular collisions. The resulting 95% Wilson confidence interval overlaps with the one observed in the dataset, thus implying that a statistically significant difference between the two groups (dataset and generated) is not supported by the data. The opposite is true for the complementary case of vehicle-pedestrian collisions. Indeed, in this scenario the estimated probability of having a collision event between a pedestrian and a vehicle has been estimated at 79%, significantly higher than the 74% of the control. Furthermore, the Wilson confidence intervals between the two groups do not overlap, thus testifying in favour of a statistically significant difference between the success probability of each group. On the topic of the more detailed test, we have performed, for each class, 50 simulations and computed the same statistics of the previous test. The left side of Figure 5.5 depicts selected results. The comparison between the two groups, for the classes vehicle-vehicle lateral low, vehicle-vehicle lateral driver low, and vehicle-pedestrian medium, proves that the generative model outperforms significantly the random generation method. Indeed, for each of these three classes, the confidence interval derived from the empirical results of the network does not overlap with the randomly generated scenarios, thus proving the model's effectiveness. For the other three classes the network is not capable of consistently outperforming the control group. Indeed, the empirical likelihood does not outperform the one estimated from the control group, indicating that further refinements are still required. The complete results of the second, more specific testing procedure are depicted in Table 5.3. Finally, we considered as our last performance evaluation metric the number of events generated per macro class (vehicle-vehicle collision/vehicle-pedestrian collision) over a fixed number of 100 simulations conditionally on the same generation procedure outlined for the first statistical test proposed. In detail, we sampled uniformly at random a collision class of the kind vehicle-vehicle (or vehicle-pedestrian for the other 100 simulations) for each of the 100 simulations and asked the network to generate the corresponding collision scenario. Then we computed the total number of collision events witnessed over the total number of simulations and compared it with the control group. Table 5.2.1 recounts the

	Random			GAN		
	p	lb	ub	p	lb	ub
V-V lateral driver low	0.067	0.063	0.071	**0.171**	0.127	0.227
V-V lateral low	0.021	0.018	0.022	**0.062**	0.040	0.092
V-V lateral driver medium	**0.165**	0.159	0.171	$*$	$*$	$*$
V-P low	**0.425**	0.417	0.433	0.182	0.145	0.225
V-P medium	0.225	0.218	0.232	**0.468**	0.416	0.522
V-P high	**0.090**	0.085	0.095	$*$	$*$	$*$

Table 5.3. Wilson confidence intervals and associated probabilities.

results of the evaluation procedure. As it is possible to appreciate, over a fixed number of simulations the GAN surpasses the control, proving significantly more effective in generating collision events, up to more than 1.5 times the baseline for pedestrian collision events.

5.2.2 Influence of class imbalance on the learned generator

As Table 5.3 conveys, the learned model fails the statistical tests for both the class vehicle-vehicle lateral driver medium and vehicle-pedestrian high. We suggest that a cause of the network's difficulty in generating such events may be found in the sampling strategy employed at training time. Indeed, by sampling the classes uniformly we avoid overfitting the critic which in turn leads to the divergence of the losses. At the same time, however, by removing the inherent bias of the dataset, we are providing the model with an implicit bias regarding the variance within each class. Nevertheless, the topic of learning from unbalanced data in GANs is still a flourishing area of research and it is complex to generalize findings among datasets. Furthermore, as we have seen in the introductory exposition detailed in Chapter 4, the convergence of a generative architecture is strongly dependent on the loss. In turn this implies that findings that we might consider valid for a given error measure might not be generalizable to other losses. A discussion on these difficulties may be found in [5].

Chapter 6

Conclusions

This thesis work aimed to present a scalable framework for the generation of criticality scenarios within the Simulation of Urban MObility SUMO. To this end, we began with the definition of a benchmark road network consisting of four incoming lanes joined at a regulated intersection. Incoming vehicles from each lane can violate the right of way of both pedestrians and vehicles thus causing collision events. Such events may differ substantially based on several factors so the necessity for a classification system arose naturally. In this direction, we propose a literature-based methodology that employs the MAIS3+ score, which is the globally accepted metric in the context of collision gravity estimation. We employed the proposed scheme to construct a representative dataset, and trained a generative adversarial network to generate initial conditions leading, in probability, to the observed risk events. An extensive statistical evaluation of the obtained model has been performed, proving that the GAN-based approach consistently generates a variety of collision scenarios, therefore demonstrating the capabilities of the approach pioneered by the present work.

6.1 Future research

A variety of possible research directions branch from this thesis work. Starting with the proposed road network, a possible generalization of this research may consider different topologies, with the addition of multiple lanes or traffic lights. The typology of vehicles may also be expanded to allow for trains, buses, and cyclists to be included, thus requiring a more detailed classification scheme. On the other hand, it is possible to consider the present topology as fixed. In this case, a natural path to follow would be the generation of a balanced benchmark dataset to assess the performances of different GAN architectures and stabilize their training.

Appendix A

A.1 GAN

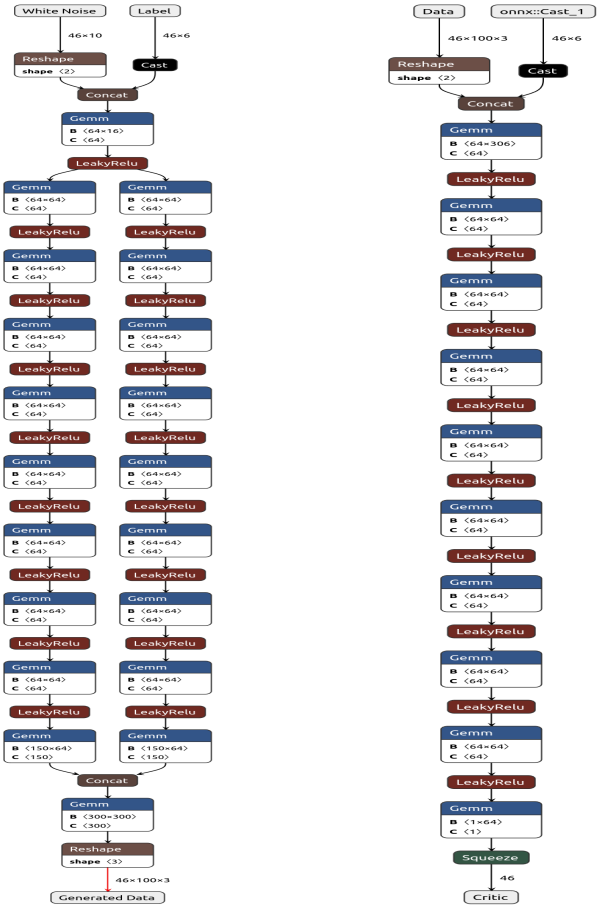


Figure A.1. *On the left*: generator, *on the right*: critic.

A.2 Wilson score intervals

The problem of evaluating the performance of the generative network proposed in the presented work has been addressed from a statistical viewpoint. In particular, we have provided Wilson score 95% confidence intervals to understand whether the data generated by the GAN had significantly different properties from the control. The Wald score intervals are approximate binomial proportion confidence intervals for the probability of success calculated from the outcome of a series of success-failure experiments. That is, an interval estimate of a success probability p when only the number of experiments (collision events) n and the number of successes n_s (the required collision class) are known. In this direction, a Wison interval can be derived by starting with the normal approximation of the binomial:

$$z_\alpha = \frac{p - \hat{p}}{\sigma_n},$$

where z_α is the standard normal interval corresponding to the desired confidence $1 - \alpha$, p is the actual probability and $\hat{p}$ is the one estimated from the trials.

$$\sigma_n = \sqrt{\frac{p(1-p)}{n}},$$

therefore, by combining these two formulas we obtain a quadratic formula in p whose two solutions are the upper and lower bounds of the confidence interval:

$$\left(1 + \frac{z_\alpha^2}{n}\right) p^2 - \left(2\hat{p} + \frac{z_\alpha^2}{n}\right) p + \hat{p}^2 = 0,$$

indeed:

$$p \in (w^-, w^+) = \frac{1}{1 + \frac{z_\alpha^2}{n}} \left(\hat{p} + \frac{z_\alpha^2}{2n} \pm \frac{z_\alpha}{2n}\sqrt{4n\hat{p}(1 - \hat{p} + z_\alpha^2}\right).$$

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] S. Amari. A theory of asaptive pattern classifiers. *IEEE Transactions*, 1967.

[3] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan, 2017.

[4] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. Sumo - simulation of urban mobility: An overview. In *SIMUL 2011, The Third International Conference on Advances in System Simulation*, 2011.

[5] Y. Chen, D. J. Kempton, and R. A. Angryk. Examining effects of class imbalance on conditional gan training. In L. Rutkowski, R. Scherer, M. Korytkowski, W. Pedrycz, R. Tadeusiewicz, and J. M. Zurada, editors, *Artificial Intelligence and Soft Computing*, pages 475–486, Cham, 2023. Springer Nature Switzerland.

[6] R. Decther. *Learning while searching in constraint-satisfcation problems.* University of California, Computer Science Department, Cognitive Systems Laboratory., 1986.

[7] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[8] D. A. Edwards. On the kantorovich–rubinstein theorem. *Expositiones Mathematicae*, 29(4):387–398, 2011.

[9] J. Erdmann. Sumo's lane-changing model.

[10] J. Erdmann and D. Krajzewicz. Sumo's road intersection model.

[11] D. for Transport. Estimating clinically seriously injured (mais3+) road casualties in the uk. 2015.

[12] I. J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[13] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[14] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans, 2017.

[15] Herbert, RObbins, Siegmund, and David. A convergence theorem for non negative almost supermartingales and some applications,. *Optimizing Methods in Statistics*, 1971.

[16] C. R. Jones and B. K. Bergen. People cannot distinguish gpt-4 from a human in a turing test. In *https://arxiv.org/abs/2405.08007*, 2024.

[17] M. Jovanovic and M. Campbell. Generative artificial intelligence: Trends and prospects. *Computer*, 55(10):107–112, oct 2022.

[18] C. Jurewicz, A. Sobhani, J. Woolley, J. Dutschke, and B. Corben. Exploration of vehicle impact speed – injury severity relationships for application in safer road design. *Transportation Research Procedia*, 14:4247–4256, 12 2016.

[19] A. U. Kemloh Wagoum, M. Chraibi, J. Zhang, and G. Lämmel. Jupedsim: an open framework for simulating and analyzing the dynamics of pedestrians. 12 2015.

[20] S. Krauß, P. Wagner, and C. Gawron. Metastable states in a microscopic model of traffic flow. *Physical Review E*, 55(5):5597, 1997.

[21] B. Laura and et al. Traffic simulation for all: a real world traffic scenario from the city of bologna. In *Modeling Mobility with Open Data: 2nd SUMO Conference 2014*. Springer International Publishing, 2015.

[22] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[23] M. Mackay. Injury and collision severity. *12th Stapp Car Conf.*, pages 207–219, 02 1968.

[24] C. Marco, C. Casetti, and G. Gagliardi. Vehicular traffic simulation in the city of turin from raw data. In *IEEE Transactions on Mobile Computing 21.12*, 2021.

[25] W. Niebel and C. Dalaff. Dlr – institute of transportation systems, traffic management research. 01 2008.

[26] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[27] F. Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1962.

[28] J. D. States. The abbreviated and the comprehensive research injury scales. *STAPP Car Crash Journal. 13*, 1969.

[29] P. Wagner, G. Flötteröd, R. Nippold, and Y.-P. Flötteröd. Simplified car-following models. In *Transportation Research Board 91st Annaul Meeting*, Januar 2012.