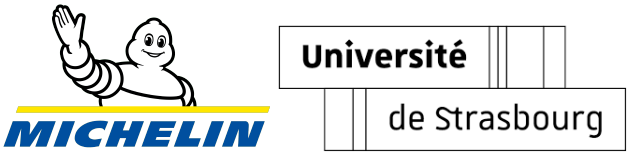

Exploration and evaluation of highly constrained neural networks.

FONSECA HINCAPIE Diana Sol Angel

University of Strasbourg
Faculty of Mathematics and Computer Science
Master of Scientific Computing and Information Mathematics

Under the supervision of :
MEUNIER Raphaël
OUBARI Fouad

February-August , 2024

Contents

List of Figures

List of Tables

Acronyms List

- NN : Neural Network.
- DNF: Disjunctive Normal Form.
- SC : Sous-creux (in French).
- AE : Autoencoder.
- VAE : Variational Autoencoder.

Acknowledgments

My supervisors, Raphaël Meunier and Fouad Oubari, for their invaluable guidance and mentorship. Their support and encouragement have empowered me to grow not only technically but also personally throughout my internship. I am also deeply thankful to the data science team for their companionship and for making this experience both enjoyable and enriching.

Abstract

In this report, we will delve into an emerging area of deep learning that focuses on developing advanced tools that enable domain experts to incorporate their prior knowledge into the training process of deep learning models. In an industrial context these machine learning models must guarantee to operate within distinct boundaries and constraints that are specified by an expert, in the case of Michelin one of it's advantages on the market comes from it's knowledge and technical expertise that has been developed through the years, we will be interested in the experimentation and testing on how to encode that domain knowledge in the form of constraints. We will start by revising the current state of the art and then we will be testing on some application cases to see how the addition of constraints affects the models' performance and it's results.

Résumé

Dans ce rapport, nous approfondirons un domaine émergent de l'apprentissage profond qui se concentre sur le développement d'outils avancés permettant aux experts sur un domaine d'intégrer leurs connaissances du métier dans des modèles d'apprentissage profond. Dans un contexte industriel, ces modèles d'apprentissage automatique doivent garantir un fonctionnement dans des limites et des contraintes distinctes spécifiées par un expert. Dans le cas de Michelin, l'un de ses avantages sur le marché vient de ses connaissances et de son expertise technique développées au fil des années. On va explorer la façon d'encoder les connaissances de ce domaine sous forme de contraintes. Nous commencerons par présenter l'état actuel de l'art, puis nous testerons sur certains cas d'application pour voir comment l'ajout de contraintes affecte les performances des modèles et leurs résultats.

1 Introduction

Michelin, a global leader in tire manufacturing, the group has accumulated throughout the years a rich domain knowledge. This knowledge is essential in ensuring that their products meet the highest standards. The R&D teams, particularly the simulation team, employs digital simulations tools that are crucial for decision-making and that help to understand and enhance the performance of future tire models, but the associated computation times and data preparation of these procedures can slow down the virtual design process. The idea to accelerate these simulations is to use deep learning methods. By integrating this expertise into deep learning models we expect they can leverage this domain knowledge effectively improving model performance but also ensuring compliance with industry-specific requirements.

In recent years, we have noticed how the the field of deep learning has made significant advances, leading to breakthroughs across various industries. However, the usage of these advanced models in industrial settings often requires them to operate under specific constraints and boundaries defined by domain experts. This ensures that the models not only perform well but also adhere to fixed standards and safety regulations. The primary focus of this internship is to explore and evaluate highly constrained neural networks, particularly in the context of Michelin's operations.

This report will begin with a comprehensive review of the current state of the art in constrained neural networks. We will examine various techniques and methodologies that have been proposed to integrate domain constraints into neural networks training. Following this theoretical overview, we will present and apply particularly the MultiplexNet architecture and the inclusion of domain and monotonicity constraints to two application cases to demonstrate how these approaches can be applied. The first case explores the addition of constraints to a generative model for tire components, while the second case focuses on constraining tire damage. Each case will be then analyzed to understand the impact of incorporating constraints on model performance and results.

Through this exploration, our goal is to test the integration of domain knowledge into deep learning models. We aim to ensure that these reliable models can be broadly utilized without concerns about non-compliance with physical laws and predetermined constraints. By investigating and applying constrained neural network, we want to demonstrate their potential for enhancing model reliability and effectiveness.

2 Presentation of the Hosting Company : Michelin

Michelin is one of the world leaders in tire manufacturing and whose activity domains range from directly working with tires, around tires and beyond tires. It has a history of over 130 years of innovation. Michelin is present in over 175 countries all over the world and counts over 132.000 employees 121 production sites and an annual turn over,in 2022, of 28.590 MDS EUR, 698 M EUR of which were invested in R&D according to the Corporate presentation Michelin by the DCEM (Engagement & Brands Corporate Direction) on April 2023.

Michelin orients its services towards human progress to offer a better way of advancing for all humanity, and mainly focusing on three main fields of activity also known as sources of sustainable growth defined as it follows :

1. **Activities with tires** : Activities that focus on the tire product like usage, typologies and solutions for all kind of mobility.
2. **Activities around tires** : Activities that focus on mobility intelligence (Leveraging mobility data), Connected services platform (Developing a network of services connecting fleet customers and trusted service providers), fleet management (Developing heavy duty tire sales and tire services. Improving fleet profitability and performance, Supporting the transition to zero-emission mobility).
3. **Activities beyond tires** : Activities that focus on leveraging Michelin's unique expertise in new markets like Experiences, Hydrogen, Medical, 3D Metal Printing, Engineered Polymers and flexible composite materials.

The company has a focus on an environmentally responsible future based on an equilibrium between people, economic and financial performance and the planet. It's environmental politics aim at controlling all pollution risks and to decrease the environmental footprint of tires until a neutral impact is achieved.

Michelin aims at becoming a reference in the field of inclusion and diversity, where the values of equal chance, gender balance, identity and disability are promoted.

Michelin wants by 2050 to be recognized as a leader in innovation and whose actions are decisive to help humanity advance and conquer new frontiers.

2.1 AI and Data Science at Michelin

For it's simulation tasks Michelin relies heavily on numerical simulations mainly based on the Finite Elements method, but in recent years it's R&D department has started to shift it's focus leveraging the new opportunities that the advancements in AI has brought. Michelin is one of the first to integrate artificial intelligence in its development processes, this is mainly possible thanks to data availability that is a product of decades of measurements and statistical analysis and well established computational resources. The applications of AI within Michelin are vast and range from different fields, for example :

- The Integration of Machine Learning into physical models.

- Generative models to design mechanical parts and how their are used in factories.
- Prediction of the performance of a tire's parts.
- Data augmentation and defect integration for defect detection.
- The development of a GPT likeChatbot for internal usage *Aurore*.
- Among others.

The work carried out at the R&D department is based on exploring and testing new approaches as the state of the art evolves with the purpose of pushing innovation forward.

2.1.1 Data Science Team

The Data Science team takes on a diversity of projects. Among their lines of work there are the development, and implementation of prediction models regarding tire performance, training of employees of other teams in the subject of statistics and data processing and support them. The projects and models that are developed by the team will be exploited by customers and internal partners.

3 Context

My internship took place at Michelin's Research and Development Center at Ladoux, I was a part of the R&D department, specifically the Simulation (SIM) department where I joined the Data Science team. My main objective was to explore and evaluate constrained models applied to various use cases of interest and whose results must meet performance criteria that are physically realistic. The idea was to study how well the integration of constraints helps to improve a model's performance and its overall results.

4 Constraint Deep Learning Models

In recent years, artificial neural networks and in particular deep learning models have enabled advancements in various fields like engineering, medicine, healthcare, agriculture, environment, etc. A recent rise in popularity in deep learning models is mainly credited to the high computational resources and the enriching data that the world has to offer nowadays

We start by reminding that the representational ability of neural networks also known as their capacity to be widely used in many contexts. Deep learning models extract features from raw data automatically, a concept also known as feature learning. They are able to learn hidden features and relations between attributes and later leverages those same relations to predict results. This well established by means of the universal approximation theorem :

The *universal approximation theorem*, states that neural networks are capable of arbitrarily accurate approximations of a given target function.

Formally, for any given function f in a target class and $\epsilon > 0$, there exists a network $\mathcal{N}$ such that $\|f - \mathcal{N}\| < \epsilon$. In topological language, the theorem states that a set of networks is dense in the class of the target function. In this sense, the closure of the network defines the range of functions it network can represent.

As universality provides the expressive power of a network structure, studies on the universal approximation theorem have attracted increasing attention. Examples include the universality of multi-layer perceptrons (MLPs) the most basic neural networks [Cyb89]. The universality of these models demonstrates their power in approximating a wide range of functions and their potential for solving complex tasks.

which include trend in the advancement of deep learning is to provide expressive tools that allow domain experts to encode their prior knowledge into the training of neural networks. We expect deep learning models to achieve a high level of precision close to machine precision, while also assuring an accurate representation of physical phenomena and real-life results in the context of engineering and science, for that, one possible way is to incorporate additional constraints. Although the field of optimization has long been concerned with developing methods to enforce constraints, the focus is starting to shift nowadays mainly over data-driven models.

As discussed before in many fields, the expert has a certain knowledge about the system that they are modeling, which must be enforced in an exact manner for example in the context of convective processes for climate modeling, constraints can be used to enforce conservation laws, or equations of state within machine precision, (double precision, 52 significant bits) to be exact is the standard across operational climate models without degrading performance. This can be necessary for providing adequate safety guarantees, compliance or for improving the system's efficiency : training a system with less data, or less computation costs.

Domain knowledge : Also called background knowledge refers to specific information about the problem we are dealing with. As shown in Figure 1, it is defined by an expert and then included in a model to enhance it's overall performance. This information can be in the form of relevant features, concepts, taxonomies, rules-of-thumb, logical constraints, probability distributions, mathematical distributions, causal connections and so on.

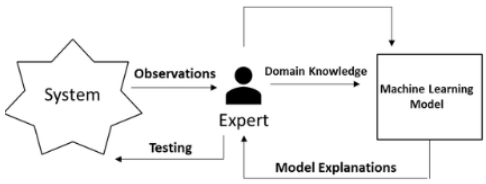

FIGURE 1 : Domain Knowledge inclusion

4.1 Importance of domain knowledge inclusion into Deep Learning Models

The interest in prior knowledge arises when considered as an important source of information that can for example augment existing data and/or increase significantly a model's performance.

4.2 Difficulties in inclusion of domain knowledge into Deep Learning Models

There are many difficulties that may arise with the inclusion of domain knowledge into deep neural networks. First there are many issues purely related to implementational aspects.

- There is no standard framework for translating logical constraints to neural networks.
- Any form of logical constraint is not differentiable.
- While numerical constraints are introduced into the loss function or as regularization terms, it is not straightforward and sometimes not very intuitive.

4.3 Types of domain knowledge

4.3.1 Physical constraints

Physical constraints include :

- Monotonicity constraints : In contexts of finance (house pricing, credit scoring, insurance risk) or the medical field (medical diagnosis, patient medication) or wear prediction for pieces or process among others.
- Material properties : Energy conservation, normalization and material symmetries.
- Physical properties in the form of equations that describe a variety of physical phenomena.

4.3.2 Geometrical constraints

They concern the desired shape and physical structure that an industrial piece should respect and are relied to the domain and the form of the desired output or prediction.

Remark : The constraints types we will explore throughout the use cases are geometrical, domain-related and monotonicity constraints.

5 State of the Art Domain-Knowledge Inclusion

There are three main types of domain knowledge that can directly be translated into alterations of either the input data of a deep network, the loss-function, the model's structure or it's parameters. We will present for each type a review of some of the stat-of-the-art approaches that were

thoroughly explored during the internship or that were found and might be of interest for further developments in the field.

We define the following parameters for constructing a deep neural network model **M** : First we have some given data **D**, a structure (π) which compounds the number and type of layers and neurons per layer and the parameters (θ) (weights and biases) a learner **L**. The training of the model is based on the learner **L** that searches to optimize the loss by performing an iterative estimation of the parameters (θ) given the model structure π.

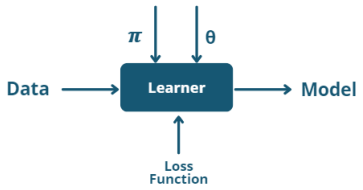

FIGURE 2 : Construction of a deep neural model

5.1 Transforming the input data

By transforming directly the input data of a neural network the domain-knowledge is primarily in symbolic form (for example, logical relations that are known to hold in the domain). The idea is based of the fact that if a data instance could be described using a set of attributes that not only includes the raw feature-values but also includes more details from the domain, then we can simply build a standard deep network from these new features. In Figure 3 we can see a scheme of how we can introduce background knowledge into deep neural network by transforming data, *trans* is a transformation block that takes input data Data, background knowledge and outputs transformed data Data* that is then used to construct a deep model using a learner.

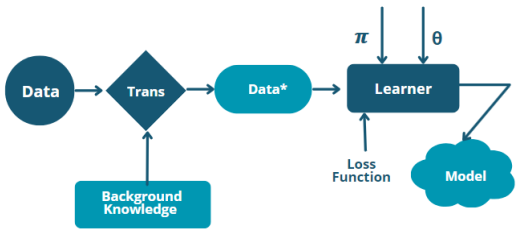

FIGURE 3 : Introducing background knowledge into deep neural network by transforming data

This type of approach won't be further explored as during the throughout search for compiling this stage-of-the art there were not any special approaches in this line that were interesting for our case. We found various approaches for example from the paper *Using Domain-Knowledge to Assist Lead Discovery in Early-Stage Drug Design* [Das+21] where they propose the combination of deep neural networks and Inductive Logic Programming (ILP), a new discipline which investigates

the inductive construction of first-order clausal theories from examples and background knowledge. That allows the use of symbolic domain-knowledge to explore the large space of possible molecules in their context. There are many applications of this case but none concern specifically the fact that the constraints should be respected at machine precision or for industrial or safety reasons.

5.2 Transforming the loss function

Another way of incorporating domain-knowledge into a deep neural network is by introducing *penalty* terms into the loss function that penalize the non-respect of the constraints imposed by domain-knowledge as shown in Figure 4. The optimizer used for model construction then minimizes the overall loss that includes the penalty terms. We will explore some of the state-of-the art methods that apply this approach and particularly the inclusion of monotonicity constraints.

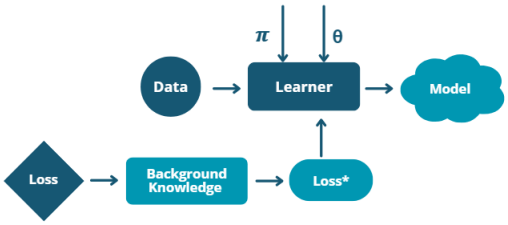

FIGURE 4 : Introducing background knowledge into deep neural network by transforming the loss function **L**

5.2.1 Physics Informed Neural Networks

Physics-informed neural networks [RPK19] (PINNs) are a type of universal function approximators that can embed the knowledge of any physical laws that govern a given data-set in the learning process, and can be described by partial differential equations (PDEs).

PINNs approximate PDE's solutions by training a neural network to minimize a loss function ; Which includes terms reflecting the initial and boundary conditions along the space-time domain's boundary and the PDE residual that encodes the governing physics equations at selected points in the domain.

Given an input point in the integration domain, they work by integrating the mathematical model into the network and reinforcing the loss function with a residual term from the governing equation, which acts as a penalizing term to restrict the space of acceptable solutions. These neural networks are constrained to respect any symmetries, in-variances, or conservation principles originating from the physical laws that govern the observed data, as modeled by general time-dependent and nonlinear partial differential equations. This simple yet powerful construction allows us to tackle a wide range of problems in computational science and introduces a potentially transformative technology leading to the development of new data-efficient and physics-informed learning machines.

PINNs aim to apply these physical constraints by revisiting the construction of "custom" activation and loss functions that are tailored to the underlying differential operator. They encourage the enforcement of physical laws but do not guarantee them.

5.2.2 Physics-constrained 3D Convolutional Neural Networks for Electrodynamics

In this paper a Physics-Constrained Neural Network (PCNN) approach to solving Maxwell's equations is proposed for the self-fields generated by relativistic charged particle beams [SP23]. This PCNNs satisfy hard constraints by construction and soft constraints are also implemented in the loss function, regarding the physics constraint, they are built into the structure of the PCNNs.

Some advantages of this approach are that the value of the loss is smaller while achieving better test data results and a much smaller violation of the physics constraint. What we can take from this article is how the Maxwell equations are portrayed in the logic of the code which can be extrapolated for other cases.

5.2.3 Monotonicity constraint imposition approaches

The monotonic dependence of the outputs of a neural network on some of its inputs is a crucial inductive bias in many scenarios where domain knowledge dictates such behavior. This is especially important for interpretability and fairness considerations. In a broader context, scenarios in which important can be found in finance, medicine, physics, and other disciplines.

5.2.3.1 How to Incorporate Monotonicity in Deep Networks While Preserving Flexibility ?

In this approach [Gup+19] a novel gradient-based point-wise loss function for enforcing partial monotonicity with deep neural networks. it aims at enhancing the learning process by incorporating a priori knowledge-monotonicity, to leverage domain expertise into data-driven approaches. The addition of a model-agnostic point-wise loss function that's acts as a plug-in to the standard loss and penalizes non-monotonicity. It is able to learn differentiated individual trends and produces smoother conditional curves that are important for personalized decisions, while preserving the flexibility of deep networks.

Point Wise Loss Function

A point-wise loss (PWL) function that incorporates monotonic knowledge into neural networks by altering the learning process. The objective function with point-wise derivatives which embeds *a priori* knowledge about monotonicity is inspired from finite element analysis as approximation and classes of functions, We formulate the following minimax objective function $\mathcal{L}_{mono}$ computed over x_i, $\forall i \in [1, n]$:

$$\min \mathcal{L}_{mono} = \min \left\{ \sum_{i=1}^{n} \max \left(0, -\nabla \cdot M \, f(x_i; \theta)\right) + \mathcal{L}_{NN} \right\} \tag{1}$$

where $\nabla \cdot M$ is divergence with respect to feature set $x[M]$, i.e., $\sum_j \frac{\partial f(x_i;\theta)}{\partial x[M]_j}$ $\forall j \in M$, θ are the trainable parameters, and $\mathcal{L}_{NN}$ refers to the empirical risk minimization for neural networks.

Other approaches as *Lipschitz Monotonic Networks* [NKW22] where a weight-constrained architecture is proposed with a single residual connection to achieve exact monotonic dependence in any subset of the inputs. The weight constraint scheme directly controls the Lipschitz constant of the neural network and thus provides the additional benefit of robustness, this method is guaranteed to produce monotonic dependence, and is highly expressive.

On the paper *Not Too Close and Not Too Far : Enforcing Monotonicity Requires Penalizing The Right Points* [Mon+21] where a practical scheme to enforce monotonicity in neural networks with respect to a given subset of the dimensions of the input space. The proposed approach focuses on the setting where point-wise gradient penalties are used as a soft constraint alongside the empirical risk during training.

Remark : Both the previous types of constrained models for domain inclusion in general fail to assure the respect of constraints at machine precision level, therefore we will be mostly interested in the following methods.

5.3 Transforming the model's structure

Now we can introduce constraints on the model parameters or by making a design choice of its structure as shown in Figure 5. We will mainly focus on two papers that assure the compliance of predetermined constraints at the level of precision we are interested, we will then explore them in the application cases. Additionally we will briefly introduce other approaches found that can be further inspected in the state-of-the-art annex.

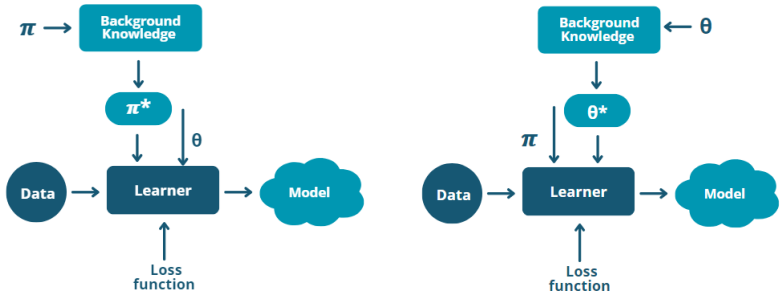

FIGURE 5 : Introducing background knowledge into deep neural network by transforming directly the model's architecture (right) or it's parameters (left)

5.3.1 MultiplexNet

MultiplexNet [Hoe+22] is an approach that represents domain knowledge as a logical formula in disjunctive normal form (DNF) which is easy to encode and to induce by human experts. It introduces a categorical latent variable that learns to choose which constraint terms to optimize directly into the output of existing learning algorithms. In this approach the constraints are compiled directly into the standard neural network's output, this allows us to ensure that any output from the

MultiplexNet network will satisfy the constraints.

Disjunctive Normal Form
Any logical formula can be compiled to DNF, a statement is in disjunctive normal form if it is a disjunction (sequence of ORs) consisting of one or more disjoints, each of which is a conjunction (AND) of one or more literals, this representation is not unique.

If we consider a data set of N i.i.d. samples and we assume that the data set was generated by some random process $p^*(x)$; and that there is some domain-knowledge about the random process, in the form of a logical formula Φ, that expresses the domain where $p^*(x)$ is feasible (non-zero). As summarized as it follows :

$$x \sim p^*(x) \implies x \models \Phi \tag{2}$$

Where $x \models \Phi$, denotes that the sample x that satisfies the formula Φ For example, if $\Phi := x > 3.5 \wedge y > 0$, and given some sample $(x, y) = (5, 2)$, we denote : $(x, y) \models \Phi$. Our objective is to approximate $p^*(x)$ with some parametric model $p_\theta(x)$ and to incorporate the domain knowledge Φ into the maximum likelihood estimation of θ, on the available data set.

Given the knowledge of the constraints Φ, we are interested in ways to integrate these constraints into the training of a network that approximates $p^*(x)$. And any sample $\tilde{x}$ from the model, $\tilde{x} \sim p_\theta(x)$, should imply that the constraints are satisfied, this is not always respected in alternative approaches.

How are constraints encoded

Constraints in MultiplexNet consist of any quantifier-free linear arithmetic formula. We denote $\tilde{x}$ the unconstrained output of a network and let g be a network activation that is element-wise non-negative (for example an exponential function, or a ReLU or Softplus If the property to be encoded is a simple inequality $\Phi : \forall x\ cx \geq b$ where c and b $\in \mathbb{R}$, it is sufficient to constrain $\tilde{x}$ to be non-negative by applying g and thereafter applying a linear transformation f such that :

$$\forall \tilde{x} : cf(g(\tilde{x})) \geq b.$$

In this case, f can implement the transformation $f(z) = sgn(c)z + \frac{b}{c}$ where sgn is the operator that returns the sign of c. By construction we have :

$$f(g(\tilde{x})) \models \Phi \quad (2)$$

While conjunctions serve to restrict the space permitted by the network's output, disjunctions serve to increase the permissible space. If we have two terms ϕ_1 and ϕ_2 in $\phi_1 \vee \phi_2$ there exist three possibilities : $x \models \phi_1$ or $x \models \phi_2$ or $(x \models \phi_1) \wedge (x \models \phi_2)$. Which means that any unconstrained network output can be transformed to satisfy some term ϕ_k.

The idea behind **MultiplexNet** comes form the introduction of multiples transformations each one designed to model each term ϕ_k and it's branching technique of disjunctions that allows the network's output layer to be viewed as a multiplexer in a logical circuit that permits for a branching of logic. If $h_1(\tilde{x})$ represents the transformation of $\tilde{\tilde{x}}$ that satisfies ϕ_1 and $h_2(x)$ represents ϕ_2 then

we know the output must also satisfy $\phi_1 \vee \phi_2$ by choosing either h_1 or h_2. If we assume that the domain knowledge Φ is expressed as :

$$\Phi = \phi_1 \vee \phi_2 \vee \ldots \vee \phi_k \quad (6)$$

if h_k is the branch of MultiplexNet that ensures the output of the network $x \models \Phi$ then it follows by construction that $h_k(\tilde{x}) \models \Phi \quad \forall k \in [1, ...K]$.

Loss Function

MultiplexNet introduces a latent Categorical variable k that selects among the different terms $\phi_k, k \in \{1, \ldots, K\}$. The model then incorporates a constraint transformation term h_k conditional on the value of the categorical variable.

$$p_\theta(x) = p_\theta(h_k(x)|k)p(k) \quad (3)$$

A lower bound on the likelihood of the data can be obtained by introducing a variational approximation to the latent Categorical variable k. This form of the variational lower bound (ELBO) can be expressed as :

$$\log p_\theta(x) \geq \mathbb{E}_{q(k)} \left[\log p_\theta(h_k(x) \mid k)\right] + \log p(k) - \log q(k) \quad (4)$$

$$:= \text{ELBO}(x) \quad (5)$$

The categorical variable can be marginalised out leading to the following learning objective :

$$\mathcal{L}(\theta; x) = -\sum_{k=1}^{K} q(k) \left[\log p_\theta(h_k(x)|k)\right] + \log p(k) - \log q(k) \quad (6)$$

Remark : This approach can be equally successfully for a generative modeling task as for a discriminative task which demonstrates the universal applicability of incorporating domain knowledge into the training of networks.

Architecture

MultiplexNet provides a new way of representing domain constraints directly in the output layer of a neural network that will guarantee that all the constraints are satisfied. To achieve this we represent the domain knowledge as a logical formula in the **DNF** form by adding this additional layer. It adds to each term in the **DNF** formula a transformation, there is also the inclusion of a latent categorical variable **k** already explained before that will select the best transformation that optimizes the loss function. This way the complex domain constraints are represented and in an automated manner and the output of the neural network will satisfy those constraints.
There are some advantages of this approach, for example any sample from the model that approximated the random process should imply that the constraints are satisfied, also, any standard neural network design can be transformed into a MultiplexNet. But there is also the difficulty of the technical specification of the domain knowledge and to the practical implementation of this knowledge, because expressing the domain knowledge in this form of logical formula may not always be possible.

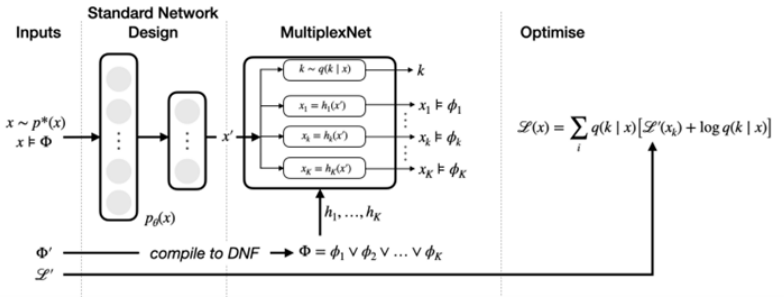

FIGURE 6 : MultiplexNet Architecture

5.3.2 Constrained Monotonic Networks

In this approach [RS23] instead of standard fully connected or dense layers, a monotonic dense layer is proposed to build constrained monotonic networks which can approximate any continuous partially monotonic function on a compact subset. A constrained monotone fully connected layer is introduced which can be used as a drop-in replacement for a fully connected layer to enforce monotonicity.

This is done by construction, by combining the idea of constraining the weights of the fully connected neural network to have only non-negative (for non-decreasing variables) or only non-positive values (for non-ascending) variables. This idea used in conjunction with non-saturated activation functions (the original activation function and two additional monotonic activation functions constructed from it) in a neural network with constrained weights, it can approximate any monotone continuous functions.

Constrained monotone fully connected layer

Activation functions

The main idea of the construction is based in the usage of three zero centered, monotonically increasing activation functions, each applied to a part of neurons in a layer :

- The original activation function $\breve{\rho} \in A$,
- Concave upper-bounded function $\breve{\rho}$.
- Bounded function $\tilde{\rho}$.

The construction is based on generating two additional activation functions from a typical non-saturated activation function such as ReLU, ELU and SELU.
$\breve{\mathcal{A}}$ is used to denote the set of all zero-centred, monotonically increasing, convex, lower-bounded functions. Let $\breve{\rho} \in \breve{\mathcal{A}}$, then

$$\hat{\rho}(x) = -\breve{\rho}(-x)$$

$$\tilde{\rho}(x) = \begin{cases} \breve{\rho}(x+1) - \breve{\rho}(1) & \text{if } x < 0 \\ \hat{\rho}(x-1) + \breve{\rho}(1) & \text{otherwise} \end{cases}$$

The construction itself is also preconditioned on a priori knowledge of (partial) monotonicity of a multivariate, multidimensional function f.
Let $f : K \mapsto \mathbb{R}^m$ be defined on a compact segment $K \subseteq \mathbb{R}^n$. Then its n-dimensional **monotonicity indicator vector** $\mathbf{t} = [t_1, \ldots, t_n]$ is defined element-wise as it follows :

$$t_j = \begin{cases} 1 & \text{if } \dfrac{\partial f(\mathbf{x})_i}{\partial x_j} \geq 0 \quad \text{for each } i \in \{1, \ldots, m\} \\ -1 & \text{if } \dfrac{\partial f(\mathbf{x})_i}{\partial x_j} \leq 0 \quad \text{for each } i \in \{1, \ldots, m\} \\ 0 & \text{otherwise} \end{cases}$$

Given an $(m \times n)$-dimensional matrix $\mathbf{M}$ and n-dimensional monotonicity indicator vector $\mathbf{t}$, the operation $|\cdot|$ is defined assigning an $(m \times n)$-dimensional matrix $\mathbf{M}' = [M'_{ij}]$ to $\mathbf{M}$ element-wise as follows :

$$M'_{j,i} = \begin{cases} |M_{j,i}| & \text{if } t_i = 1 \\ -|M_{j,i}| & \text{if } t_i = -1 \\ M_{j,i} & \text{otherwise} \end{cases}$$

All this is then implemented in the Monotonic Dense Unit (**MonoDense class**) that uses then weight constrains, the activation functions constructed before.

Architecture

The following type-1 architecture shown in Figure 7 resembles a standard MLP type of neural network architecture, where each of the input features is concatenated to form one single input feature vector x and fed into the network, but instead of standard fully connected or dense layers, monotonic dense units are used.

For the input layer the indicator vector $\mathbf{t}$, is used to identify the monotonicity property of the input feature with respect to the output, t is set to 1 for those components in the input feature vector that are monotonically increasing, to -1 for those components that are monotonically decreasing and set to 0 if the feature is non-monotonic respectively.

For the subsequent hidden layers, monotonic dense units with the indicator vector t always being set to 1 are used in order to preserve monotonicity. Finally, depending if the task is a regression or a classification problem an appropriate activation function to obtain the final output is given.
One of the main advantages of this approach is it's simplicity we can build deep NNs with different architectures that can be built using the proposed monotonic Monodense layer. The main advantage compared to other state-of-the art methods is computational and memory complexity, there are less parameters making the NNs more energy efficient. And most importantly these networks are able to approximate any multivariate monotonic function.

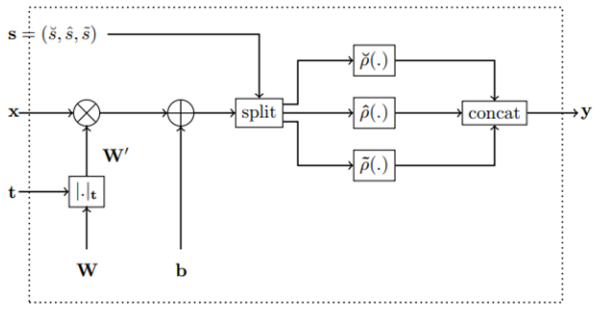

FIGURE 7 : Neural architecture type-1 using the MonoDense layer

The papers discussed in this section explore various innovative approaches to integrate constraints into neural network architectures by transforming the model's structure. There are other papers that could not be explored during my internship but that are interesting for future exploration.

In *"Deep Isotonic Embedding Network : A Flexible Monotonic Neural Network* a the Deep Isotonic Embedding Network (DIEN) is developed to handle both monotonic and non-monotonic features, ensuring monotonicity through Isotonic Embedding Units and a Monotonic Feature Learning Network.

The paper *"Enforcing Analytical Constraints in Neural Networks Emulating Physical Systems,"* proposes methods to enforce conservation laws within neural networks by integrating constraints into the network architecture or loss function, with applications in climate modeling.

The approach *RAYEN, a framework for imposing hard convex constraints on neural networks*, ensures constraint satisfaction with minimal computational overhead.

Lastly, *"DeepSaDe : Learning Neural Networks that Guarantee Domain Constraint Satisfaction"* presents an approach for enforcing semantic domain constraints in neural networks, leveraging Maximum Satisfiability Modulo Theories (MaxSMT) for training.

6 Use cases

6.1 Generation of tires' components

6.1.1 Context

This application case takes place as a part of a project which objective is the optimization of tires' components by generative design to efficiently process their conception. The idea is to build a generative model with the objective of having new tire architectures to help designers get more diversity while preserving épure coherence. The focus will be on a component known as *épure*. At

Michelin, an *épure* refers to a tire's interior cross-section, this cross-sectional view provides a detailed understanding of the tire's internal composition, enabling precise analysis and quality control during the manufacturing process. The concept of an épure is crucial for ensuring that the tire's internal geometry adheres to Michelin's standards.

Other generative models have already been tested by other colleagues in the Data Science team by implementing VAE (Variational Autoencoder) models, the main problem appears when we observe that those generations do not reflect the reality of what expect an "épure" should look like as if it was for example modeled using a CAO/CAD program, it is at this point that the need for hard constraining the assembly and contact of the pieces forming the "épure" arises. The idea is that by including constraints we will be able to ensure that the generation will be physically realistic, by having the characteristics of a real-world "épure".

FIGURE 8 : Example of an épure

6.1.2 Data description

The dataset contains a set of 279 samples obtained from simulations of tire's épures an exemple is shown in Figure 9, each sample is composed of 27 products or pieces. We will focus on the X and Y point cloud coordinates that describe each product.

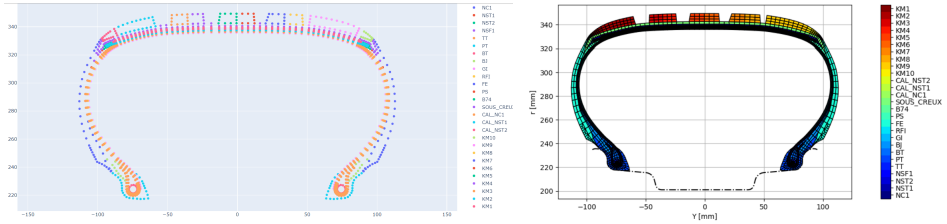

FIGURE 9 : Sample from épure data simulations

6.1.3 Methodology

After a thoroughly review of the state-of-the art, that was presented in the section State of the Art Domain-Knowledge Inclusion We first tried a segmentation approach by implementing the MultiplexNet framework but then we quickly moved to another idea.

To start we decided to test the constraint inclusion by reconstructing **KM1, KM2** shown in Figure 10 and then **KM1, KM2 and Sous creux** (under hollow in english but we will contnue to call it

Sous-creux) show in Figure 11, to measure the scalability of both methods.

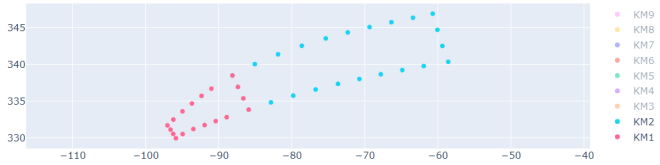

FIGURE 10 : Example samples from products to test KM1 and KM2

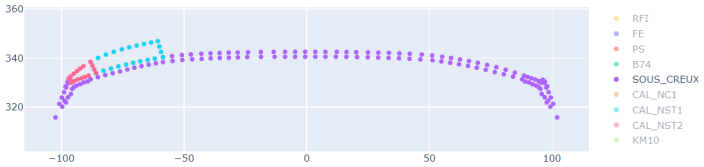

FIGURE 11 : Example samples from products to test KM1, KM2 and Sous-creux

We employed two architectures an Autoencoder **(AE)** and a Variational Autoencoder **(VAE)**.
An **Autoencoder** is used to learn efficient embeddings of unlabeled data for a given network configuration. The autoencoder consists of two parts, an encoder, and a decoder. The encoder compresses the data from a higher-dimensional space to a lower-dimensional space (also called the latent space), while the decoder converts the latent space back to higher-dimensional space. AEs excels at learning compact data representations, but the latent space is not regularized, meaning it lacks specific constraints on the distribution of points in space Z. Consequently, while proficient in compression and reconstruction, AEs cannot effectively generate new data.

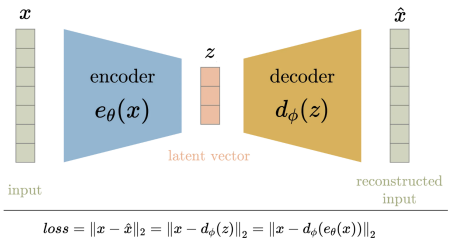

FIGURE 12 : AE Architecture - Image credit : Medium

A **Variational autoencoder** addresses the issue of non-regularized latent space in AE models that was previously discussed, and provides the generative capability to the entire space. The encoder in the AE outputs latent vectors. Instead of outputting the vectors in the latent space, the encoder of VAE outputs parameters of the priori distribution in the latent space for every input. The VAE then imposes a constraint on this latent distribution forcing it to be a normal distribution. This constraint makes sure that the latent space is regularized.

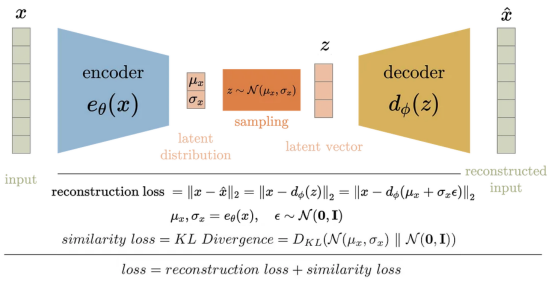

FIGURE 13 : VAE architecture - Image credit : Medium

We will now shift our focus back to our use case, initially we integrated the constrains after training. However, due to the challenges encountered, we opted to include them directly during training.

6.1.3.1 Integrating the constraints after training

Initically the focus was on the usage of the MultiplexNet framework. MultiplexNet allows encoding multiple constraints in disjunctive normal form, which could ensure multiple constraints for various components simultaneously.

We started with a standard AE/VAE model to predict the reconstruction of components. To enhance this model, we introduce an additional layer after the output layer, implementing the MultiplexNet framework. This results in the following complete architecture :

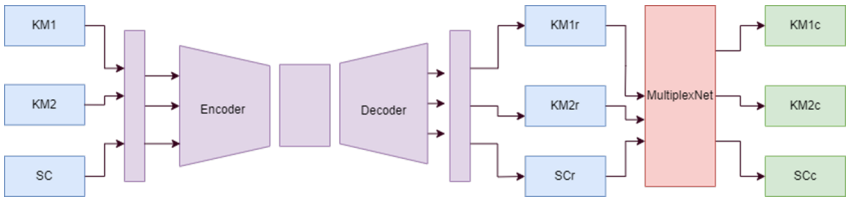

FIGURE 14 : Autoencoder model with MultiplexNet

The models were trained by calculating the loss through comparisons between the training data (blue boxes) and the constrained outputs (green boxes).

Initially the constraint was designed to be applied **deterministically** which means that it was applied after training. Subsequently, we will explore a non-deterministic method where the constraint is included and enforced during training. From the unconstrained output of the multi-modal AE/VAE model, we will outline the plan to apply the constraint with the reconstruction of KM1 and KM2 :

1. **Obtain the Unconstrained Output** The unconstrained reconstruction $\tilde{x}$ from the AE/VAE model is obtained.

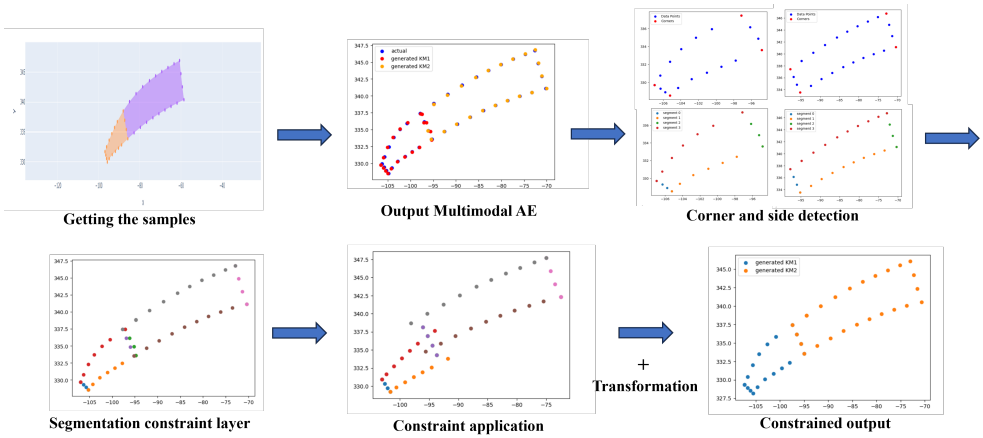

FIGURE 15 : Plan to apply constraint to KM1 and KM2

2. **Corner and Segments Detection** Then we apply the corner detection function to each product and then we get the segments delimited by them.

3. **Identify Segments of Interest** After obtaining the segmentation for each product, we identify the segment of interest where we want to apply the constraint.

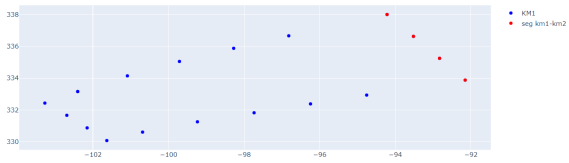

FIGURE 16 : Segment of interest for KM1

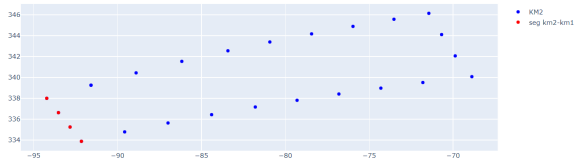

FIGURE 17 : Segment of interest for KM2

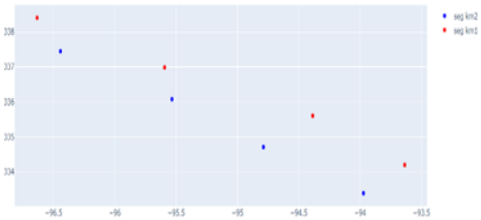

FIGURE 18 : Segments to constraint KM1 and KM2

4. **Definition of the Constraint** The constraints are defined as the point-by-point mean between the two segments shown in Figures 16, 17 and 18 as it follows :

$$\textbf{newseg} = \frac{\mathbf{KM1_{seg_2}} + \mathbf{KM2_{seg_0}}}{2}$$

This calculation generates a new segment for both products, then it will be replaced as the new common segment, allowing both products to align correctly.

5. **Apply Transformation** After defining the new segment constraint, we apply a transformation that adjusts the remaining points for each product to match the new segment, ensuring a coherent reconstruction. The mean calculation and the transformation define h_1 a constraint in the MultiplexNet framework.

6. **Obtain the Constrained Output** Finally, we obtain the constrained output $x = h_1(\tilde{x})$

The respect of **all** the contact constraints was expected, but after further analysis we decided to change our approach and not use the MultiplexNet framework because in this framework we know that the domain knowledge is expressed in the DNF as it follows : $\Phi = \varphi_1 \vee \varphi_2 \vee \ldots \vee \varphi_k$, where each ϕ_i is a constraint.

And any unconstrained output $\tilde{x}$ can be constrained to satisfy φ_k using the transformation h_k. Then, we can have that $h_k(\tilde{x}) = x \models \Phi$ which means that the constraint output x satisfies Φ but this can be achieved by respecting for example only φ_1 and φ_3 and not all $k \in [1, ...K]$ where K is the number of constraint in Φ. But our main interest is to secure the compliance of all constraints $\varphi_1 \vee \varphi_2 \vee \ldots \vee \varphi_k$ at the same time. Therefore, we chose to apply the constraint in a layer following the output layer of our multi-modal AE/VAE models.

As anticipated, the complexity of the transformation presents a problem. This method works in a deterministic approach, but continuing the training while implementing the constraint introduces issues. Specifically, segmentation problems arise when the model's reconstructions cannot be easily segmented, as shown in Figure 19.

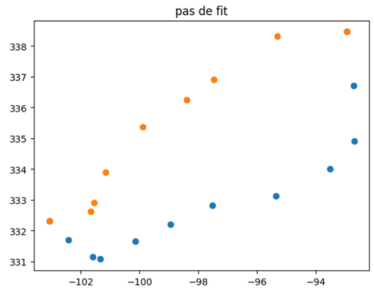

FIGURE 19 : Output where segmentation is not applicable

To get around this difficulty we tried to implement during training a verification step where we tested if segmentation was possible but then again this slowed dramatically the training process and the model had a hard time learning.

Additionally after applying a complicated transformation we evidenced by studying the area of the figure before applying the constraint and the transformation and after that the reconstruction and they were different. Without mentioning how complex the transformation became once we wanted

to reconstruct the three figures including the Sous-creux product.

These two main issues led us to abandon this approach and consider alternative methods.

6.1.3.1 Integrating the constraints during training

Now after the firs try, we kept the core idea of the contact constraint but we moved to a different method to apply the constraint by directly incorporating it during training. We also changed the previously defined method to identify and extract the segments of interest. Although all our samples have the same number of points, their construction varies, by rearranging the samples so that corresponding points are aligned at the same indices, we could easily identify the segments and apply the constraint effectively.

Reordering the data

First, for KM1, we aimed to determine the index of the maximum value. Using the previously defined segmentation, we identified the corner at index 13. Then a function to shift all points so that the maximum value consistently appeared at index 13 for all samples was applied. For KM2, no shifting was necessary as the data was already well-aligned across all samples. Finally, for the "sous-creux" products two distinct patterns were spotted in the point indexing shown in Figure 20.

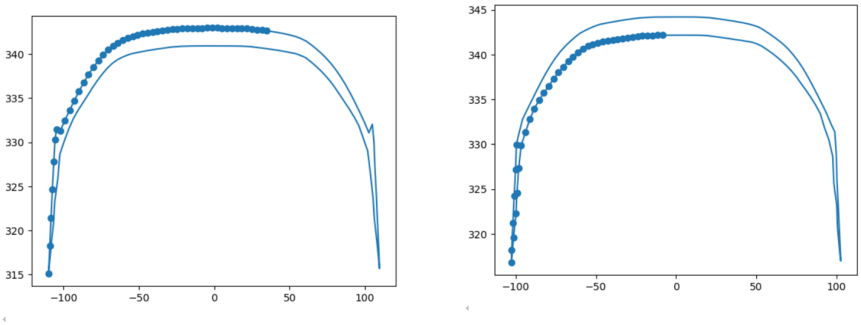

FIGURE 20 : Sous-creux indexation paths

To order the sous-creux products, a function to standardize the X and Y coordinates of all samples was defined. The strategy involved identifying the minimum point for each sample and determining its segment based on the previously defined segmentation (either segment 0 or segment 4). This allowed us to classify each sample into one of the two cases. Depending on the identified case, we then rearranged the segments to ensure consistent ordering across all samples.

After arranging the data we realized there was an issue that arose because the Sous-creux samples had previously been resampled to ensure the same number of X and Y points across all samples. Therefore, we artificially resampled the data to align the key points with the constraints from KM1 and KM2, interpolating the remaining points. This ensured that all samples for KM1, KM2, and the Sous-creux were organized and adhered to the desired constraints.

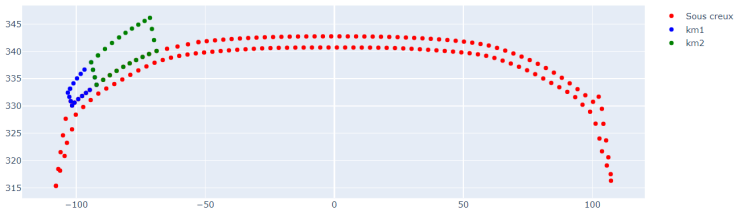

FIGURE 21 : Sample after ordering the data

Architecture

In this method, we propose adding a layer at the end of our standard VAE/AE model that defines the constraints, resulting in the following complete architecture :

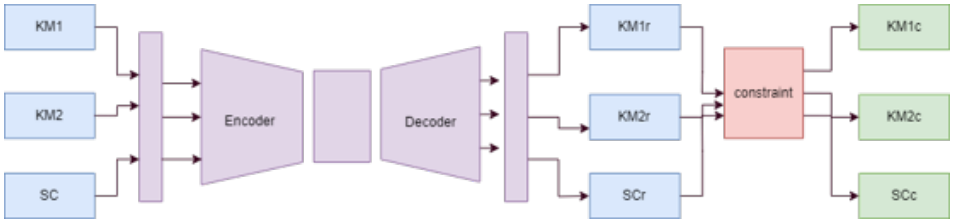

FIGURE 22 : Variational autoencoder model with constraints

Training for the whole model was done so that loss was calculated by comparing the training data (blue boxes) and the constrained outputs (green boxes) respectively.

Constraint Explanation

In this approach, we will focus on generating KM1, KM2, and the Sous-creux. From the unconstrained output of the multi-modal AE/VAE model, we obtain the unconstrained outputs $\mathbf{KM1}_u$, $\mathbf{KM2}_u$, and $\mathbf{SC}_u$.

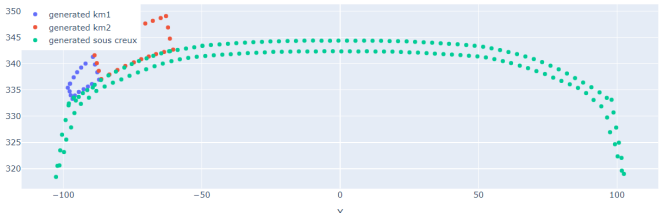

FIGURE 23 : Unconstrained reconstruction obtained from the standard multi modal VAE model

After reordering the data, we can now easily identify the indexes of the points belonging to the segments of interest in Figures 24, 25 and 26, for constraining KM1 and KM2 with the sous-creux, as they are consistent across all samples. In the constraint layer, we will extract these segments :

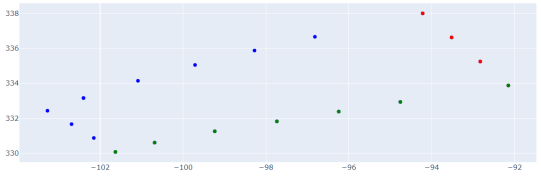

FIGURE 24 : Segments of interest for KM1

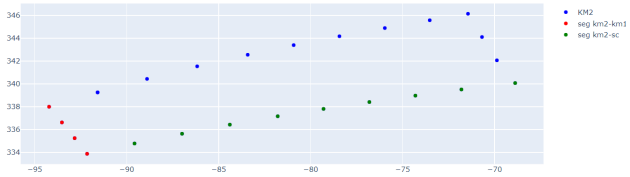

FIGURE 25 : Segments of interest for KM2

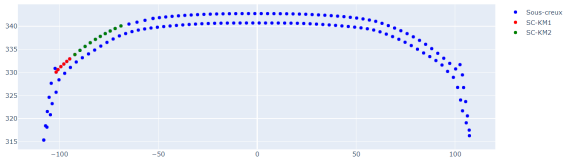

FIGURE 26 : Segments of interest for Sous-creux

And calculate their mean, point by point as it follows :

1. $\mathbf{newseg_1} = \frac{\mathbf{SC_{seg_{42}}} + \mathbf{KM1_{seg_1}}}{2}$
2. $\mathbf{newseg_2} = \frac{\mathbf{SC_{seg_{41}}} + \mathbf{KM2_{seg_1}}}{2}$
3. $\mathbf{newseg_3} = \frac{\mathbf{KM1_{seg_2}} + \mathbf{KM2_{seg_0}}}{2}$

From the specific segments of interest we obtain the results shown in Figure 28.

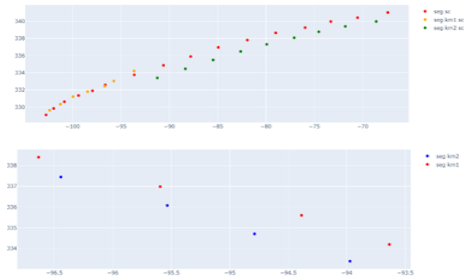

FIGURE 27 : Segments to constrain KM1, KM2, and the Sous-Creux

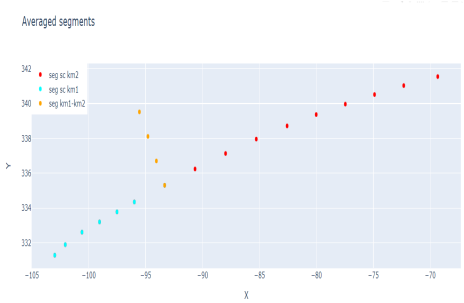

FIGURE 28 : Averaged segments of interest

We then replace those segments with the calculated means as the new segments. But this time, however, we are not concerned with transforming the unconstrained points. The constraint application is done during training where the constrained outputs are compared to the training data.

6.1.4 Results

For the first approach where we applied the constraint after training, we explored a deterministic and a non-deterministic approach where we pre-trained the model to get the reconstruction and then continued the training with the constraint. However, only the deterministic addition of the constraint for generating KM1 and KM2 was completed, due to the difficulties discussed earlier.

6.1.4.1 Integrating the constraints after training

The following results display constrained reconstructions from a pre-trained multi-modal AE, trained with a learning rate of 1×10^{-3} over 3000 epochs, with a deterministic constraint applied at the end of the pre-trained model.

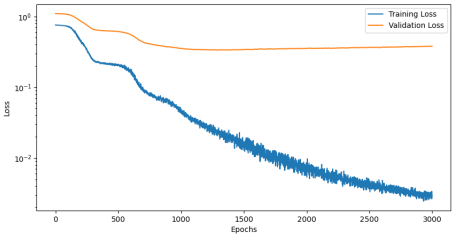

FIGURE 29 : Loss evolution

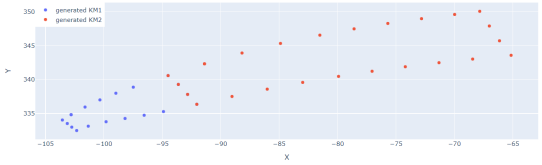

FIGURE 30 : Multi modal AE pre-trained with deterministic constraint 1

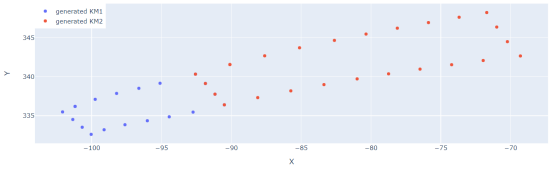

FIGURE 31 : Multi modal AE pre-trained with deterministic constraint 2

6.1.4.2 Pre-training the model and then incorporating the constraint

For this approach, we pre-trained the multi-modal AE model, followed by continued training with the constraint at a very small learning rate (1×10^{-7}) and a reduced number of 500 epochs. This allowed the model to smoothly apply segmentation to the reconstructions from the unconstrained standard model. The results obtained are the following :

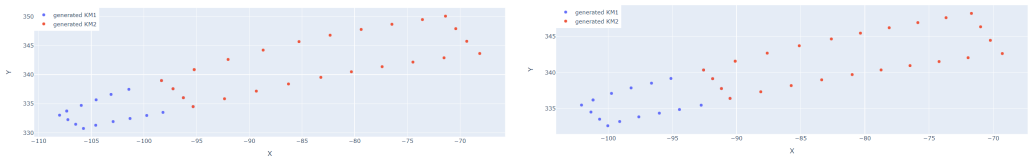

FIGURE 32 : Constrained reconstructions from AE model

The approach of training the model from scratch with the constraint included was also testes. As anticipated, it presented significant challenges, resulting in the model struggling to learn effectively. This method was quickly rejected and we moved on the next we approach.

6.1.4.3 Integrating the constraints during training

Now we present the results from both reconstruction and generation using a VAE model for KM1 and KM2, and for KM1, KM2, and the Sous-Creux. The goal was to evaluate the effectiveness of adding constraints and to conduct a *diversity study*. As mentioned earlier, we want to generate new and varied forms of products to assist épure designers in the conception phase.

Constrained reconstructions of KM1 and KM2 : For the following results we trained an AE model using the previously defined architecture with a learning rate of 1×10^{-4} over 2000 epochs.

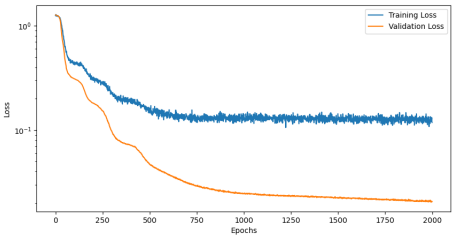

FIGURE 33 : Loss evolution

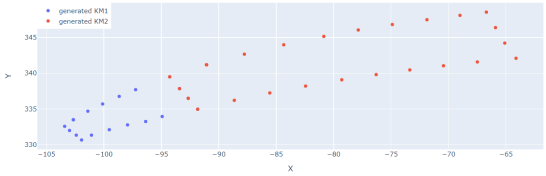

FIGURE 34 : Constrained reconstruction for KM1

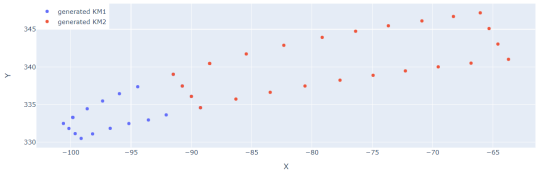

FIGURE 35 : Constrained reconstruction for KM2

Constrained reconstructions of KM1 and KM2 and the Sous-creux : We will now take interest in a multi modal VAE the idea is to carry out further studies for the generations from the latent space, for this we will first examine the results from an unconstrained model trained with a learning rate of 1×10^{-4} and 2000 epochs :

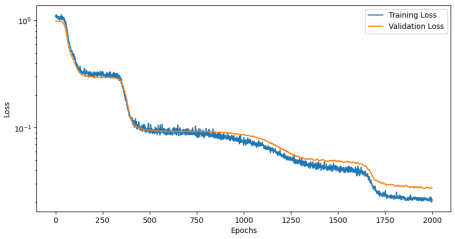

FIGURE 36 : Loss evolution

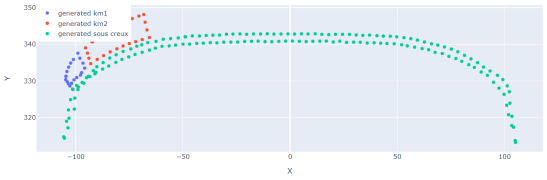

FIGURE 37 : Unconstrained reconstruction 1

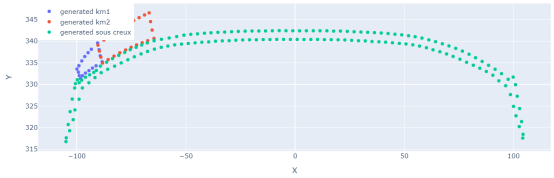

FIGURE 38 : Unconstrained reconstruction 2

And now when adding the constraint to this model trained with a learning rate of 1×10^{-3} and 1500 epochs we get the following reconstructions :

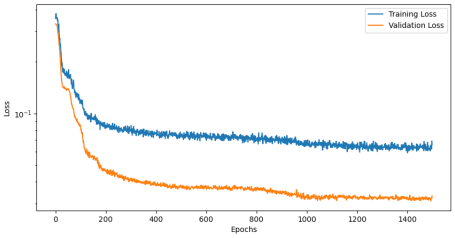

FIGURE 39 : Loss evolution

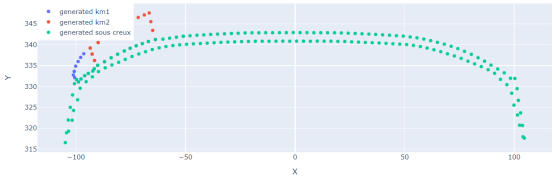

FIGURE 40 : Constrained reconstruction 1 from multi-modal VAE model

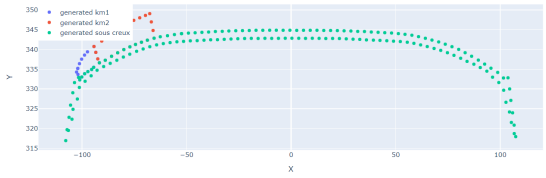

FIGURE 41 : Constrained reconstruction 2 from multi-modal VAE model

Remarks : The reconstructions of the VAE model successfully adhere to the specified contact constraints that were incorporated. This demonstrates that the model effectively integrates the constraints into the reconstruction process, ensuring that the output not only captures the underlying structure of the input data but also respects the desired realistic connectedness.

Furthermore, even if the constraint is defined simply, this approach validates the potential of incorporating constraint mechanisms into deep learning models to enhance their performance and applicability in real-world scenarios.

6.1.5 Diversity Analysis of Product Generations

In this section, we will conduct various studies to measure the diversity of the products by comparing the training dataset with the reconstructions made by the VAE model and the generated samples. We will begin by focusing on optimizing the generations from the VAE model.

6.1.5.1 Generations from VAE's Latent Space

Sampling from a VAE's latent space enables the generation of new data that is similar to the data seen during training. We will explore how to effectively sample from the latent space to produce diverse and high-quality generations, examine the diversity and fidelity of the generated samples, and compare these results with the original training dataset and reconstructions. This analysis will provide insights into how well the VAE model captures the underlying data distribution and its potential for generating innovative new product designs.

To effectively sample from the VAE for generating new data, the idea is to sample from the prior distributions while avoiding under sparse regions in the latent space. We achieve this by analyzing the latent space to determine the range and distribution of the latent vector for each dimension. Specifically, we identify the interval where 90 percent of the values lie for each dimension and sample from these intervals to decode the encoding. The idea is to ensure that new generations are within a plausible range of the learned latent space, assuring constrained realistic and diverse samples.

We will show some examples of generations from the latent space of the VAE. These images demonstrate the model's ability to create new data points by sampling from the latent space. Each sampled latent vector is decoded to produce a reconstruction of KM1, KM2 and the Sous-creux, demonstrating the diversity and variability of the generated outputs in Figure 42.

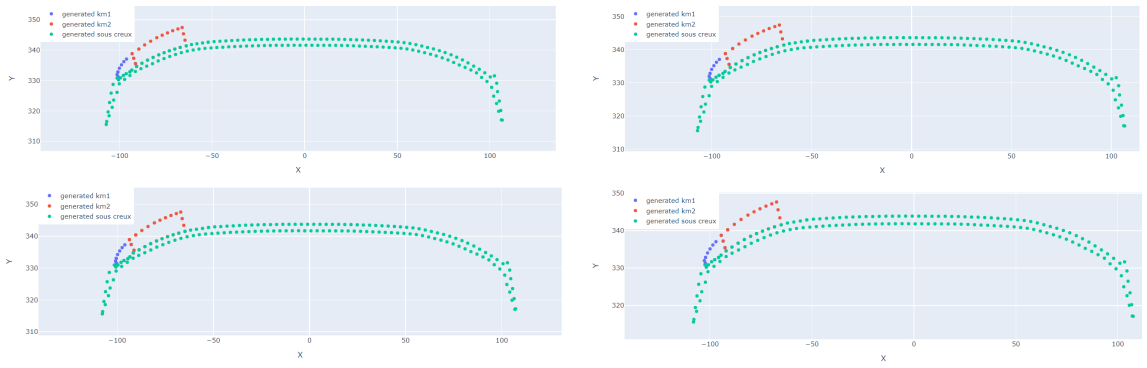

FIGURE 42 : Generations from VAE latent space illustrating the diversity of the generated images.

Remarks : From the generations we can point out that all generations seem to respect the contact constraints and are coherent to what we expected.

To carry out our analysis, we used a representative sample of 100 from each dataset : the training set, the VAE reconstructions, and the generations from the VAE latent space obtained as described before. We will conduct various studies on these samples.

6.1.5.2 Area study

We will begin by examining the average area of the products in the training datasets, as well as the reconstructed data using an unconstrained and a constrained model and the generated products. For that we will employ the following box-plots :

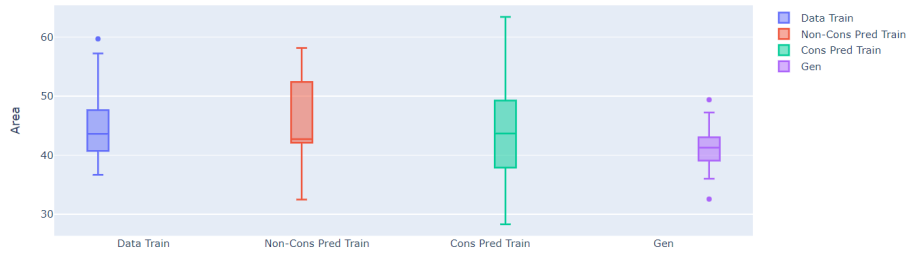

FIGURE 43 : Comparison of Areas with Training Data for KM1

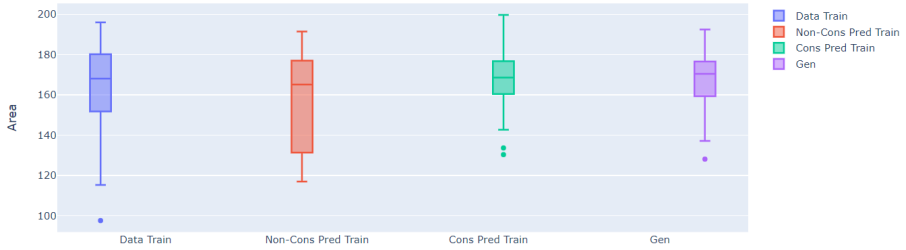

FIGURE 44 : Comparison of Areas with Training Data for KM2

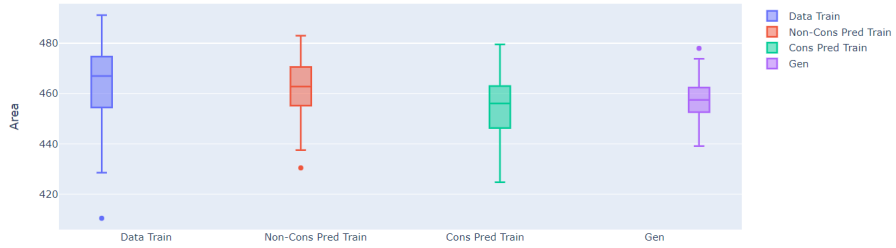

FIGURE 45 : Comparison of Areas with Training Data for SC

Remarks :

For **KM1** the mean areas are close, the results for the unconstrained model (Non-Cons Pred Train) closely aligns with the training data, indicating accurate reconstruction, while for the constrained model it maintains the general characteristics but shows slight variability. And finally the generated samples exhibit a wider range of areas which reflects a greater variability.

For **KM2** we can see that the unconstrained model reconstructs the areas closely to the training data, as before the constrained model shows similar alignment with minor variations. And the generated data displays a broader range, highlighting the model's ability to produce diverse forms.

For the **Sous-creux**, both for the unconstrained and constrained models the areas are similar to the training data, while for the generated samples they show more variability, indicating the potential for generating diverse new forms with different areas.

In overall for all products, the unconstrained models closely match the training data. The constrained models introduce slight variability but maintain the overall characteristics. The generated samples exhibit greater variability, which can be beneficial for generating diverse forms.

6.1.5.3 Products generation study

Using UMAP, a general-purpose manifold learning and dimensionality reduction algorithm, we visualize the 2D projections of the training, reconstruction, and generation data samples for each product. We will inspect these visualizations and examine examples from the distributions to observe the generated products, particularly focusing on their proximity to or deviation from the distributions of the training and reconstructed points.

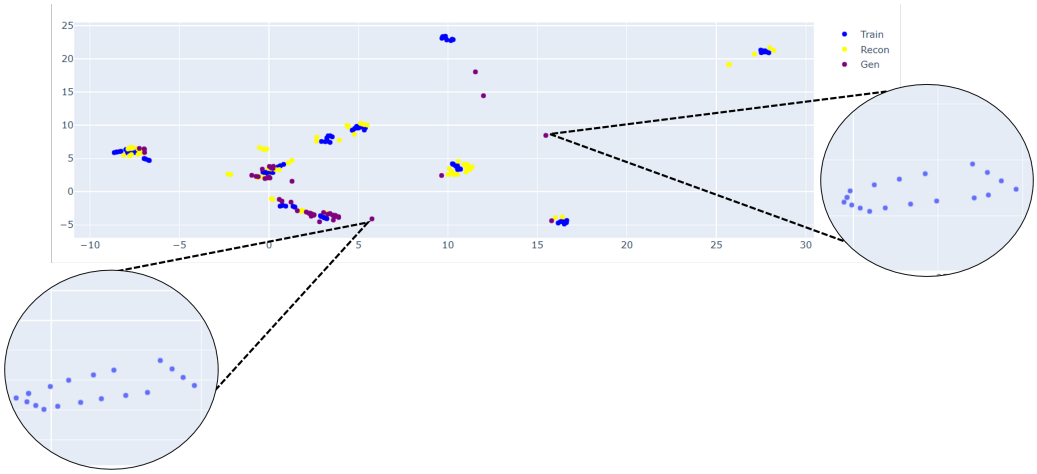

FIGURE 46 : KM1 reduction distribution

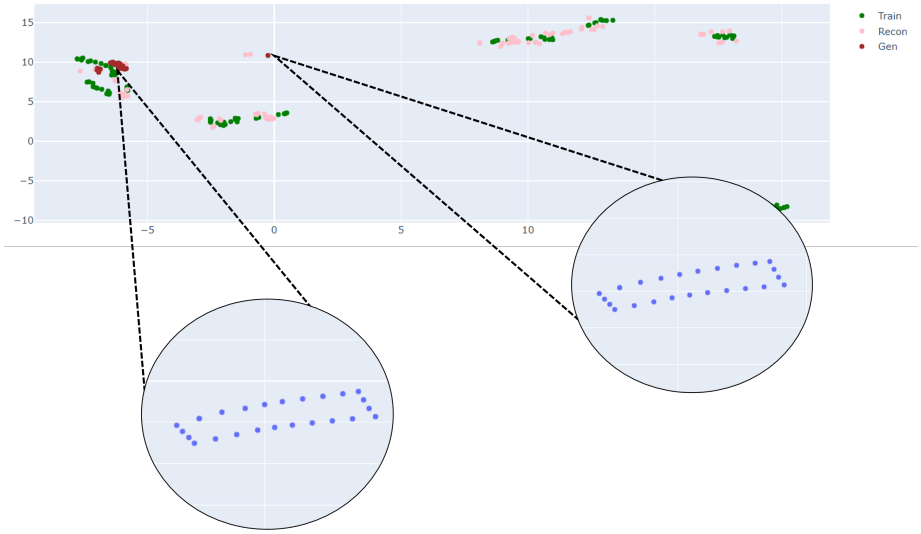

FIGURE 47 : KM2 reduction distribution

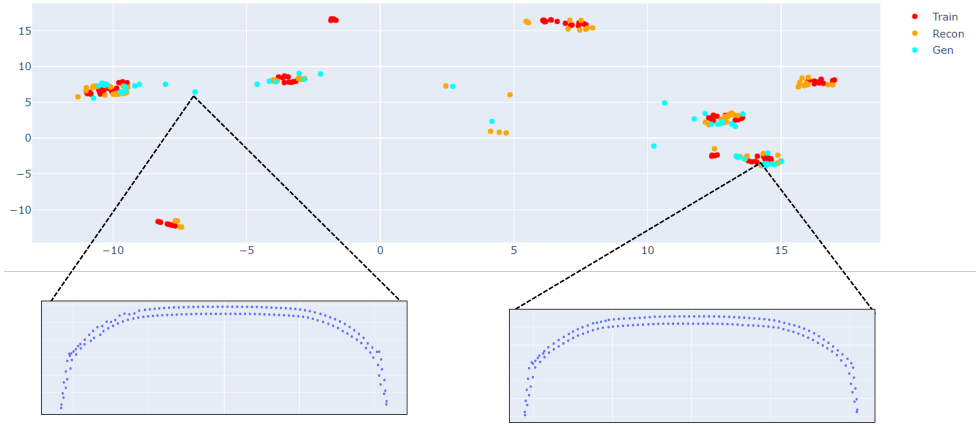

FIGURE 48 : Sous-creux reduction distribution

Remarks : Overall, we observe that for all products, the generated samples closely resemble the training and reconstruction data when they are near these distributions. However, when examining samples that deviate further from the training and reconstruction data, we notice that while their forms differ, they still maintain the overall expected shape for each product generation.

6.1.5.4 Data values distribution study

We aimed to examine the distributions of points by comparing the samples from the training, reconstruction, and generated products. This comparison will help us assess how closely the generated products resemble the training and reconstruction products.

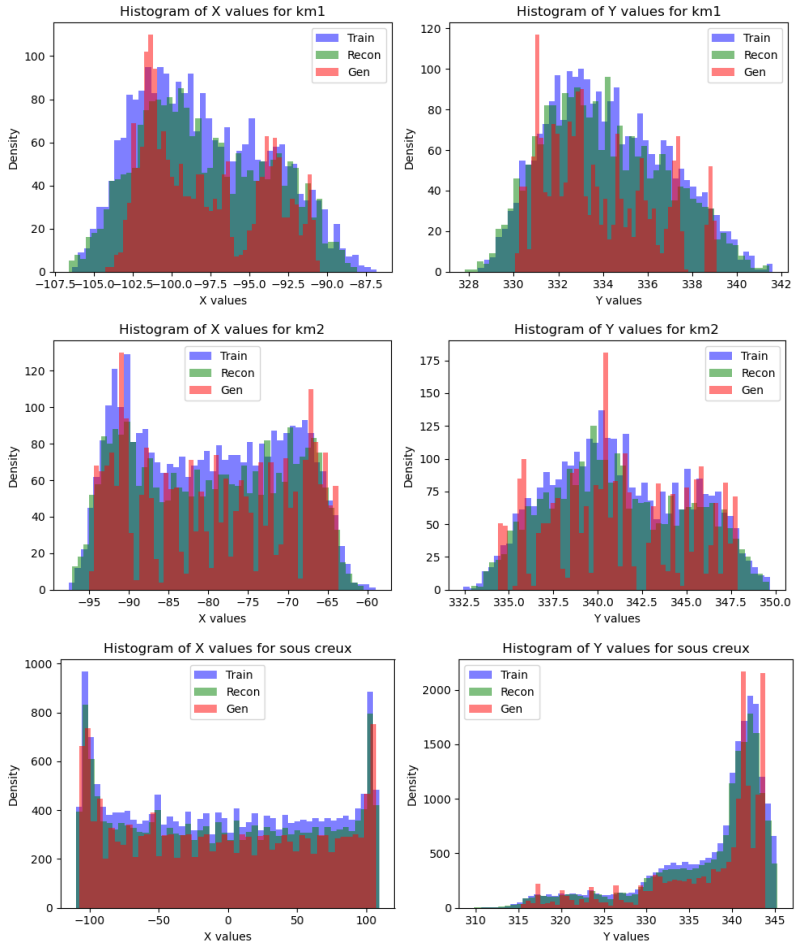

FIGURE 49 : Distributions of X an Y for each product

Remarks :

For **KM1** the histograms for KM1 show a similar distribution pattern between the training set and the VAE reconstructions, indicating that the VAE model is effectively capturing the original data distribution. The generated samples also follow the general shape of the training set but may show some deviations and the distribution shape differs a little from the expected one.

For **KM2** the distributions reflect a good match between the training set and the reconstructions. The generated samples seem to approximate the training distribution, but there may be differences in certain regions.

For the **Sous-creux**, the histograms indicate that the VAE reconstructions align well with the training set. The generated samples follow the overall trend of the training set distribution but

might exhibit some deviations.

6.1.5.5 Diversity study

The strategy for testing the diversity of the generated products involves using a classification model. We obtained randomized samples from the training set, the reconstructions from the constrained VAE, and the generated data using the previously described strategy, then by strategically choosing the samples from each group that are close to each other when projected into a 2D space as shown in Figure 50. From that we can remark that most of the generation lay in an specific region. They are then labeled as [0, 1, 2] respectively. The idea was to train different classification models to see how accurately they can predict the labels from a test subset.

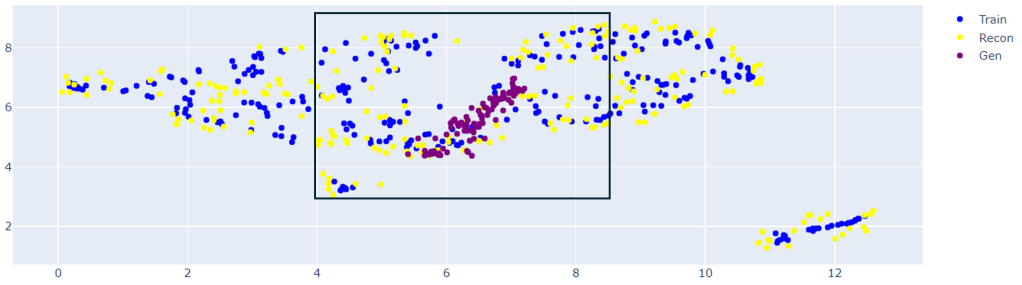

FIGURE 50 : 2D projections of the datasets

Our objective by implementing a classification model was to study the generations and see if aside from getting different and new diversities compared to the train and reconstruction datasets, some of them also resemble the original data.

However, one thing that should be kept in mind is that there is also a compromise between diversity and the quality of the generations, as show in the following results presented in Table 1.

Model	Metric	Train	Recon	Gen	Overall
SVC	Test Accuracy				1.0
	Precision	1.00	1.00	1.00	
	Recall	1.00	1.00	1.00	
	F1-Score	1.00	1.00	1.00	
Naive Bayes	Test Accuracy				0.711
	Precision	0.60	0.44	1.00	
	Recall	0.71	0.33	1.00	
	F1-Score	0.65	0.38	1.00	
Gradient Boosting	Test Accuracy				0.822
	Precision	0.92	0.61	1.00	
	Recall	0.71	0.92	0.88	
	F1-Score	0.80	0.73	0.93	

TABLE 1 : Performance metrics for SVC, Naive Bayes, and Gradient Boosting models.

For most of the classification algorithms used, the results show that they achieve almost perfectly to classify the generations. We cannot straight up conclude that the results obtained are based solely on to the fact that the generations are completely different from the other datasets. After verifying the generation we observed that some of them present a little maladjustment in a particular corner shown in Figure 51 which might influence the classification model to directly spot a generation.

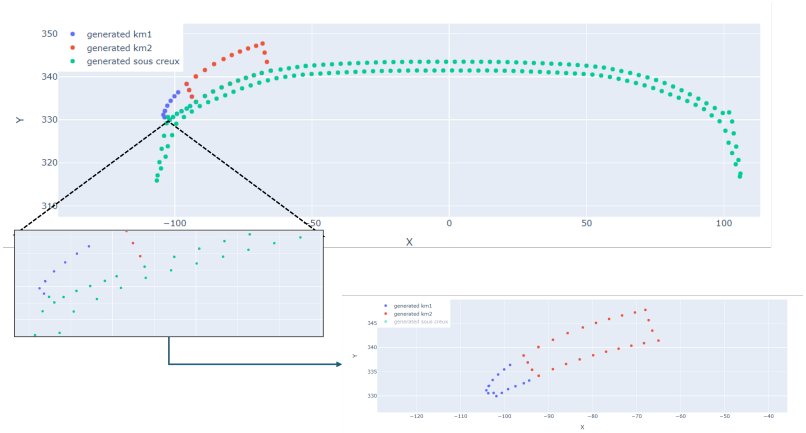

FIGURE 51 : Problem in some of the generations

After an inspection to compare the 3 datasets we obtained the following figure where we plot our datasets and color them according to their class, and we can clearly evidence that all of the generation lay on the same stripe which explains why all of the classification models are able to easily spot the generations over the two other datasets of training and reconstruction.

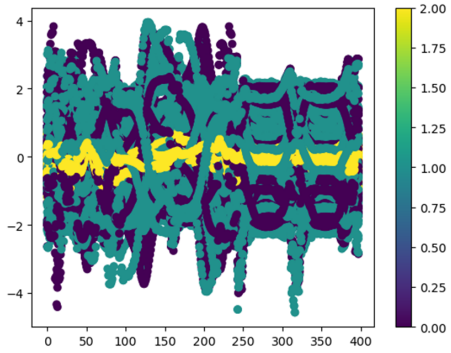

FIGURE 52 : Comparison classification data

This is a common difficulty that appears when building generative models, where generations are not generalized or diverse enough. We tried sampling from a wider interval of 70 percent instead 90 percent of our latent vectors and we got the following results for the same models :

Model	Metric	Train	Recon	Gen	Overall
SVC	Test Accuracy				0.867
	Precision	1.00	0.75	0.81	
	Recall	1.00	0.75	0.81	
	F1-Score	1.00	0.75	0.81	
Naive Bayes	Test Accuracy				0.644
	Precision	0.67	0.60	0.67	
	Recall	0.47	0.75	0.75	
	F1-Score	0.55	0.67	0.71	
Gradient Boosting	Test Accuracy				0.844
	Precision	0.88	0.73	0.92	
	Recall	0.88	0.92	0.75	
	F1-Score	0.88	0.81	0.83	

TABLE 2 : Performance metrics for SVC, Naive Nayes, and Gradient Boosting models after sampling differently.

In particular, we wanted to see which generations were well and misclassified, for that by using the Gradient boost model we get the following comparisons of samples that were well classified as generations and those who were misclassified as real data and reconstructions :

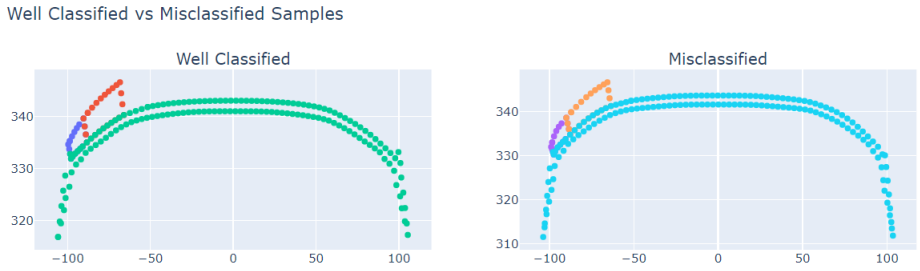

FIGURE 53 : Example misclassified as reconstruction

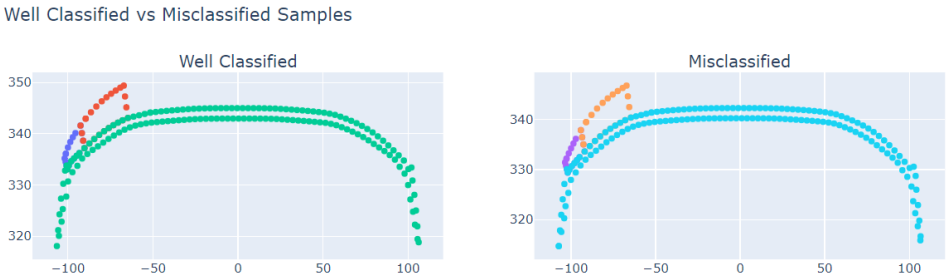

FIGURE 54 : Example misclassified as train

Remarks : From the results obtained we see that our generations are constrained, they are generally different from the train dataset, some of them might have small details as the corner issue, maybe a point that is not perfectly aligned etc. That surely affects the classification task, but once we modified the interval from which we sampled the latent vectors we got better results in terms getting diversity but also likeness for our generations. Overall, the generations are what we expected them if we consider the fact that we cannot expect a large diversity taking into account the reduced number of samples we have and the already scarce diversity among them.

6.1.6 Analysis of the results

The process of conceiving how to define and incorporate a "contact constraint" was not evident at first, but keeping segmenting was a good way to start. We started by testing a deterministic approach for the constraint application which at least now we know is not the best way and that allowed us to focus on making the constraints differentiable so that they can be incorporated directly into the training process.

The idea of applying a segmentation was also a good way to start because the core idea of the segmentation to find the segments of interest where we want the constraint to be applied was also kept for the other method. This method helped us refine our strategies and better guide our future exploration of new possibilities and ideas.

Now having learned from the previous method, we were able to implement the new approach that avoids the problematic segmentation process. This new method retains the concept of the "contact constraint" but modifies how we identify the segments of interest to apply the constraint. The results are promising for both AE and VAE models, as all reconstructions and generations respect the constraint. However, scalability remains a challenge when working with more than three products. The main challenges are identifying and ordering the segments to constrain and applying the constraints sequentially between pairs of products.

Regarding the diversity study we can conclude that these are not exactly the results we expected, we expected more generations would resemble the train dataset for the reason we already explained before, maybe with another model, architecture or a different way of constraining the results could be different. The generations are in general good regardless of the issues with the details that they present, but further exploration could have been needed to get better results.

Final comments

Ultimately, we decided to set aside this application case, mainly because after my supervisors and the designer responsible for the épures had a discussion, they learned that new and more diverse data samples were going to be available later on, this data would not have been accessible in time for me to work with it. It was concluded that the work we have done will be retained as an initial exploration. Hopefully, it will serve as a valuable starting point for future investigations.

We decided to further explore the incorporation of constraints in other cases of interest to Michelin. In the next section, we will examine a new application where we integrate domain and monotonicity constraints into deep learning models to predict tire damage.

6.2 Prediction of Damage

6.2.1 Context

Predicting tire wear is crucial for ensuring safety and optimizing maintenance schedules. Accurately forecasting the evolution and final extent of tire damage allows for better planning and prevention of potential failures. In this application case, we focused on testing the inclusion of domain and monotonicity constraints into deep learning predictive models while using synthetic data to simulate tire wear progression.

6.2.2 Data Description

The synthetic time-series dataset for predicting tire damage for a given trajectory was defined by a member of the Data Science team and has been used in multiple cases to test different approaches. It serves as a kickoff tool to test different methods and approaches in the context of damage prediction before applying them to real-world data. The time series consists of five features, and the goal is to predict the evolution of the damage (d) and its final value (d_{final}).

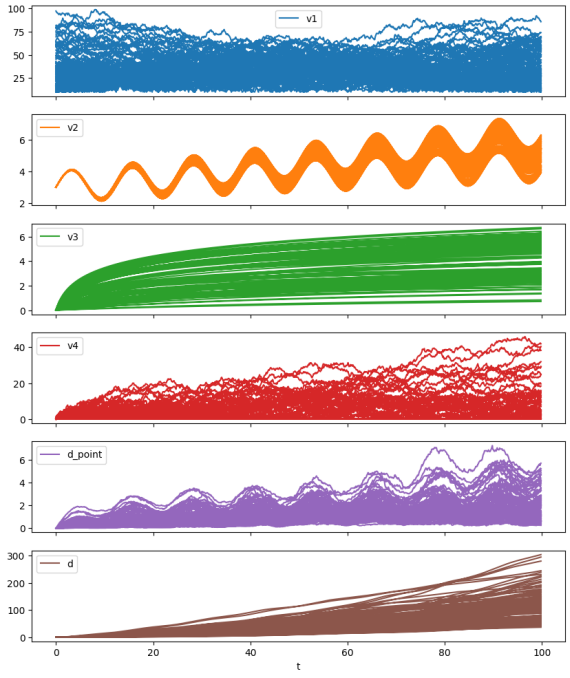

FIGURE 55 : Synthetic data features

6.2.3 Methodology

As previously mentioned we recovered the time series generator, and additionally a transformer model that predicts the rate of change of the damage at each time step and the final damage from a

team member, his focus was mainly in testing how the attention mechanism of a transformer model which is basically the model's way of making sense of the input it receives, capturing long-range dependencies in the data, could be applied to predict damage.

We focused mainly on two types of constraints. First, domain constraints, since tire damage should range between 0 and 1. Second, monotonicity constraints, which ensure that tire wear always increases over time as a reflection of the real-life behavior.

6.2.3.1 Domain Constraints

Constraint definition

For the domain constraints, we specified two different conditions : $\tilde{d} \geq 0$ and $\tilde{d} \leq 1$ where $\tilde{d}$ refers to the unconstrained wear evolution over time. When using MultiplexNet, these conditions are applied separately and encoded as an inequality constraint each. If we do not employ MultiplexNet for this simple constraint we can encode the layer with the combined constraint $a \leq \tilde{d} \leq b$. Then, in general for a constraint of this type we employ the following encoding :

$$a < x < b \quad \rightarrow \quad x = -g(-g(\tilde{x}) + k(a, b)) + b$$

Where the function $k(a, b)$ computes the correct offset for a given activation g. In this case we employ the Softplus activation function and we define $k(a, b) = \log(\exp(b - a) - 1)$.

Models for damage prediction

Once the constraints were defined, to start, we decided to test the impact of adding the MultiplexNet framework to the output layer of the existing transformer model, with the objective of evaluating how incorporating these domain constraints affect an already defined model's results.

The unconstrained transformer model yielded already good predictions for the damage evolution (d) that was defined by the model as the cumulative sum over d_{point} which is a vector that contains the rate of change at each time step for each trajectory but as show in the results on Figures 56 and 57 some trajectories do not respect the real-life results we expect regarding the domain restrictions.

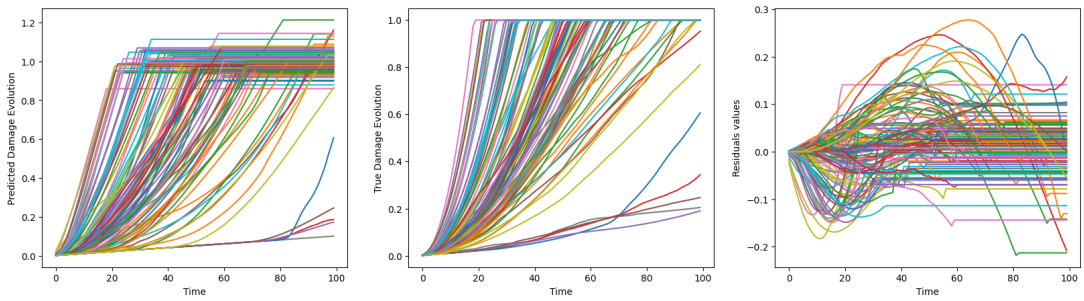

FIGURE 56 : Transformer predictions for training data

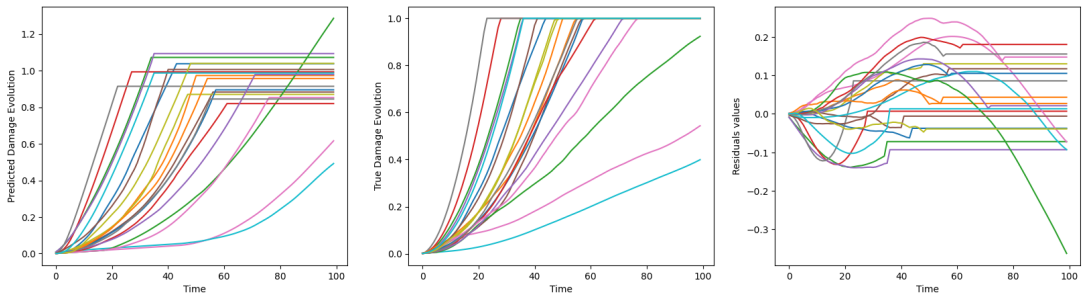

FIGURE 57 : Transformer predictions for test data

Overall the idea was to test various models suitable for the prediction task to evaluate the impact of the constraint layer on the learning process and overall predictions. The tested architectures are the following :

1. **Transformer + MultiplexNet** : This model combines a transformer architecture, known for its self-attention mechanism, with MultiplexNet, which enforces domain constraints on the output layer. The transformer layers handle the sequential data, while MultiplexNet ensures the predictions respect the given domain constraints defined before.

2. **LSTM with/without constraint** : An LSTM (Long Short-Term Memory) network, is designed to handle sequential data by maintaining long-term dependencies through its memory cells. The model is tested in two variants : one with domain constraints applied to the output layer and one without any constraints, to compare the effect of enforcing domain-specific constraints.

3. **Sliding window dense model with/without constraint** : This architecture uses a sliding window approach where a dense (fully connected) neural network processes fixed-size segments of the time series data. The model is evaluated with and without domain constraints to see how the constraints influence the learning of local patterns within each window.

4. **Standard dense model with/without constraint** : A standard dense neural network where all neurons in one layer are connected to all neurons in the next layer. The model is tested with and without domain constraints on the output layer to understand the impact of these constraints on a basic neural network architecture.

5. **Convolution model with/without constraint** : A convolutional neural network (CNN) that uses convolutional layers to extract spatial hierarchies in the data. This model is also evaluated with and without domain constraints on the output layer, to investigate how the constraints affect the model's ability to capture important features.

6.2.3.2 Monotonicity Constraints

A monotonically constrained deep learning model model is a type of model that enforces monotonicity constraints on the input features. Monotonicity means that as the value of one input feature

increases, the output of the model either always increases or always decreases, keeping other variables unchanged.

These constraints are important because they can help improve the interpretability of the model by ensuring that the model behaves in an intuitive and explainable way. In our case as mentioned before we wanted to test the possibility to enforce this type of constraints to assure the fact that a tire's damage has to monotonically increase over time.

In the transformer model, monotonicity is inherently imposed by construction. The predicted d_{point} vector, which contains the rates of change at each time step for each trajectory, accumulates to define d, the damage evolution. We also decided to build a model by implementing the Monodense layers found while carrying pout the stat-of-the art review.

6.2.4 Results

In this section, we present the results of the approaches and models tested. Our primary focus was on understanding the impact of domain and monotonicity constraints on the performance and results of different model architectures. Below, we summarize the outcomes of these experiments, highlighting key findings through different metrics.

6.2.4.1 Domain Constraints

Transformer + MultiplexNet

We first evaluated the addition of MultiplexNet to the output layer of the transformer model to impose domain constraints. The transformer model already provided good predictions of damage evolution d. However, by implementing MultiplexNet, as shown in Figures 58 and 59, we observed significant impacts on the output.

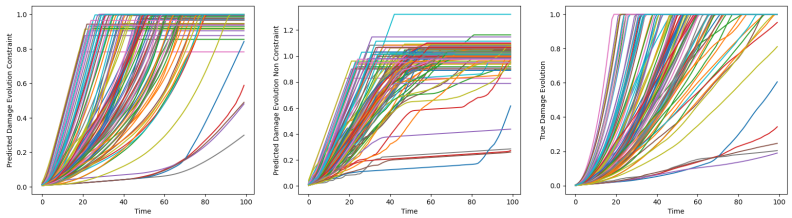

FIGURE 58 : Constrained vs unconstrained transformer predictions for train data

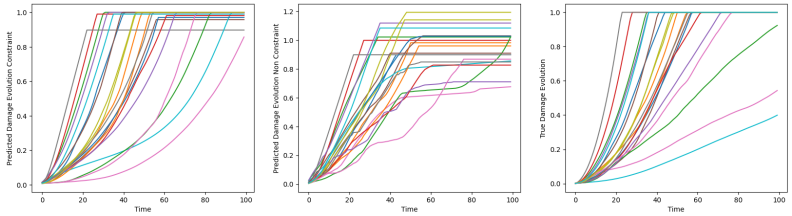

FIGURE 59 : Constrained vs unconstrained transformer predictions for test data

Remarks : For the unconstrained model, 0.2986% of the predicted trajectories violate the constraints, while the constrained model exhibits no violations (0.0%). This demonstrates that incorporating the MultiplexNet framework into the transformer model significantly enhances prediction accuracy, making it particularly valuable for tasks requiring strict adherence to domain constraints.

LSTM Model

Now for the LSTM models, each layer is conceived so that it processes the input data sequentially and updates its hidden and cell states, effectively learning temporal patterns over time. The models were trained each with a learning rate of 1×10^{-3} over 3000 epochs.

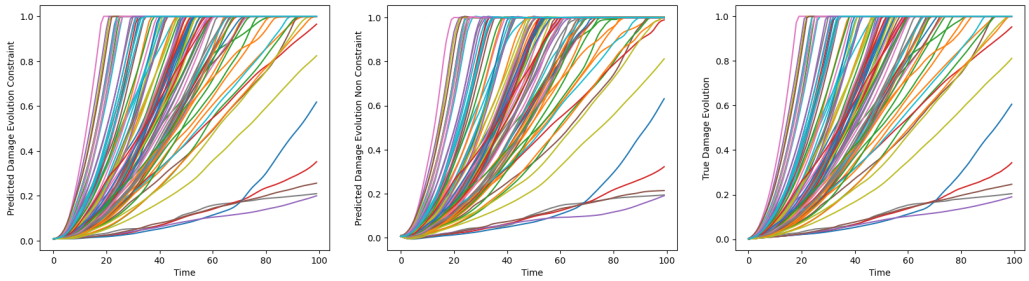

FIGURE 60 : Constrained vs unconstrained LSTM model predictions for train data

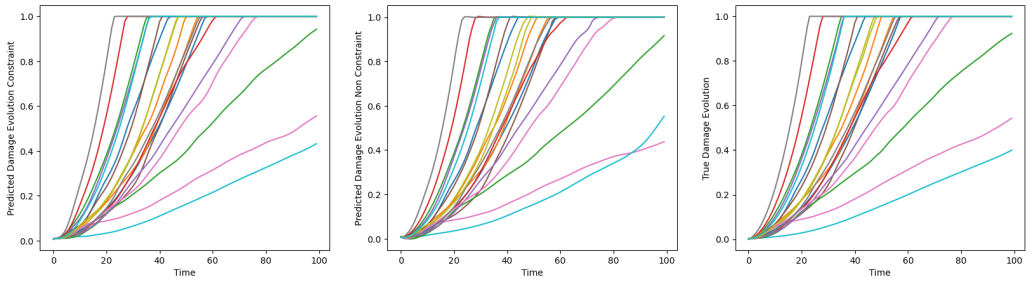

FIGURE 61 : Constrained vs unconstrained LSTM model predictions for test data

Remarks : Both models demonstrate excellent performance with highly accurate predictions. However, a closer inspection shows that in the unconstrained model, 0.4008% of the trajectories violate the constraints, while the predictions from the constrained model have no violations.
The constrained model consistently predicts trajectories that adhere to the specified constraints, ensuring that the predictions remain within the expected range throughout the entire sequence as shown in Figures 60 and 61. We remark that the constrained model's predictions are smoother and more uniform compared to the unconstrained model, the unconstrained model shows more variability and occasional deviations from the expected trajectory, which could lead to less reliable predictions.

Dense Models

Sliding window dense model

Given the nature of time series data, we decided to implement a dense model based on a sliding window approach. This model model consists of several fully connected layers, which are designed to learn complex patterns from the input windows, this allows the model to capture local temporal patterns by segmenting the time series data into smaller, overlapping windows.

Each window represents a fixed-size segment of the time series, which is then used as input to the dense network. Then, the model can make predictions based on recent data points while considering the context provided by the overlapping segments. For the training both models were trained with a learning rate of 1×10^{-3} and 800 epochs to obtain the following results.

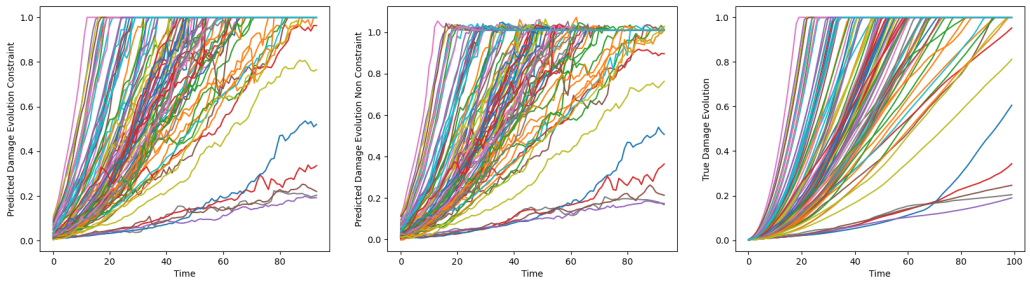

FIGURE 62 : Constrained vs unconstrained SW dense model predictions for train data

Remarks : The sliding window models demonstrate the ability to capture local temporal patterns in general, we can see clearly that for both models predictions are not smooth because the model simply learns to recognize patterns but is not aware of the overall evolution of the damage over time. In general, we can see that the addition of the domain constraints did impact the predictions but this is not an adequate model for our task.

Standard dense model

Based on the results from the sliding window dense model, we decided to implement a fully connected neural network to test if the results could improve and get smoother predictions. The models we developed comprise several dense layers with ReLU activation functions to introduce non-linearity, followed by a final output layer that generates the predictions. The model was trained with a learning rate of 1×10^{-4} over 3000 epochs, yielding the following results.

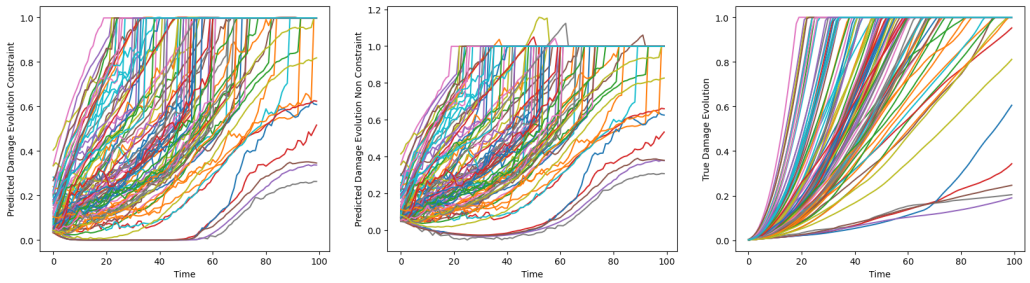

FIGURE 63 : Constrained vs unconstrained standard dense model predictions for train data

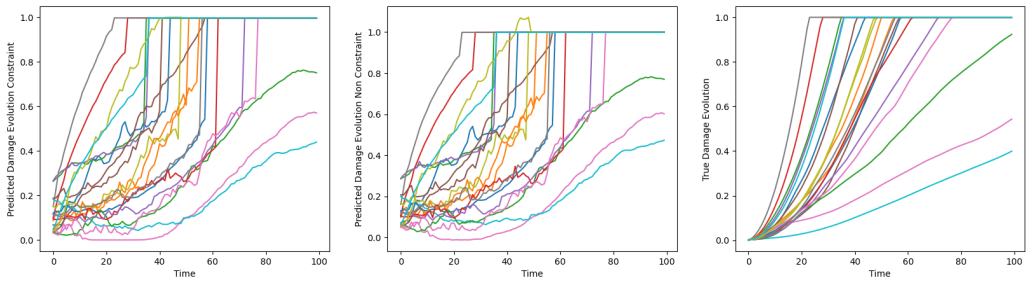

FIGURE 64 : Constrained vs unconstrained standard dense model predictions for test data

Remarks : The standard dense model produced mixed results. While the constrained model effectively respected the domain constraints, the overall performance did not significantly surpass that of the sliding window dense model. The constrained predictions were somewhat smoother and adhered to the constraints, but both the constrained and unconstrained models exhibit higher variability and less coherence in the trajectories compared to the sliding window approach. We can conclude from the general variability and the presence of sharp changes in the trajectories that the fully connected dense model may not be as well suited for capturing temporal dependencies.

Convolution Model

We decided to also implement and test convolution models for their ability to predict by effectively capturing local patterns and trends. This model applies consequent convolution layers each of which is paired with batch normalization and ReLU activation to enhance learning and performance. We wanted to test how the sequential data is handled by focusing on local dependencies and patterns. The models were trained with a learning rate of 1×10^{-4} over 2000 epochs and these are the results.

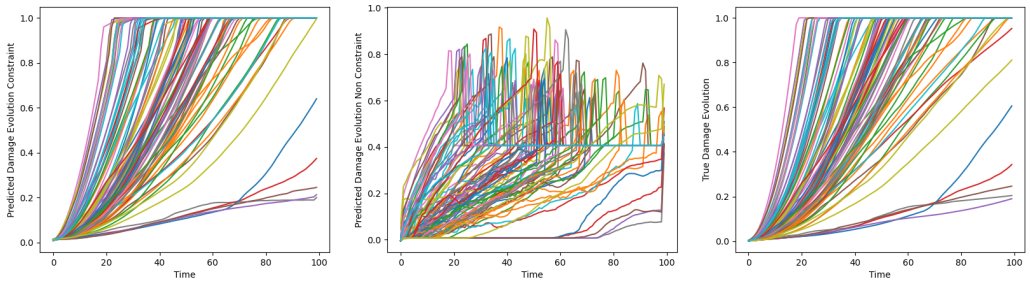

FIGURE 65 : Constrained vs unconstrained convolution model predictions for train data

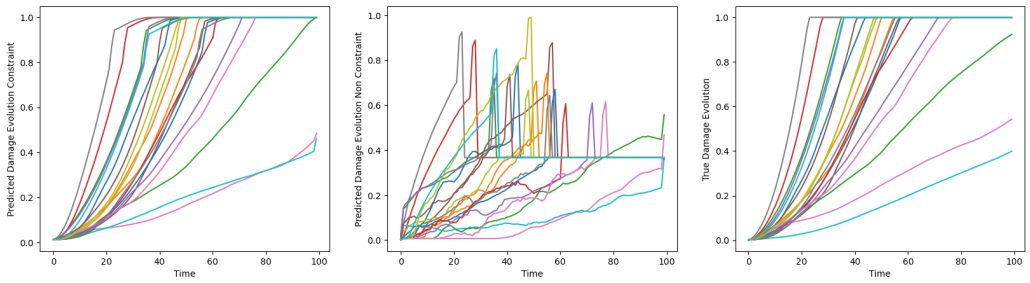

FIGURE 66 : Constrained vs unconstrained convolution model predictions for train data

Remarks : The results from the convolution models clearly demonstrate an advantage in capturing local patterns and trends within sequential data when constraints are added. The constrained model shows a noticeable improvement in the smoothness and consistency of the predicted trajectories compared to the unconstrained model. We believe this improvement is mainly due to the model's ability to focus on other important details once essential characteristics are enforced by the constraints.

In the following tables, we summarize and compare the main metrics obtained for the models tested :

Model	**MSE**	**MAE**	**RMSE**	**R2**
LSTM	3.2197	1.0895	1.7943	0.9979
Dense	23.9143	2.3130	4.8902	0.9828
Convolution	6.2355	1.2263	2.4971	0.9960

TABLE 3 : Performance Metrics for Constrained Models

Model	**MSE**	**MAE**	**RMSE**	**R2**
LSTM	6.2355	1.2263	2.4971	0.9960
Dense	26.3848	2.4132	5.1366	0.9811
Convolution	1838.6836	35.4744	42.8799	-0.1837

TABLE 4 : Performance Metrics for Unconstrained Models

Remarks : From the performance metrics summarized in the tables we can clearly evidence the significant impact of applying constraints to the models. Firstly, for the LSTM model, the constrained version exhibits lower MSE, MAE, RMSE, and a higher R2 compared to the unconstrained version. This demonstrates the effectiveness of constraints in enhancing the accuracy and reliability of the model's predictions. The fully connected dense model shows improvement with constraints, although the gains are less remarkable than those seen with the LSTM model. Finally the convolution model shows the most difference between constrained and unconstrained versions. While the constrained model performs reasonably well, the unconstrained model has a significantly higher error metrics and a negative R2 value. This suggests that without constraints, the Convolution model struggles to maintain the integrity of the predictions, leading to unreliable predictions.

6.2.4.2 Monotonicity Constraints

For this constraint application, we began by conducting a monotonicity study for each feature with respect to d (damage evolution), we derived the monotonicity indicators needed to inform the model.

The initial models we implemented were fully connected models, in which we replaced the standard dense layers with **Monodense layers** that incorporate the monotonicity indicator. The Monodense layers were applied to ensure that the output respects the monotonicity property, which is critical for accurately modeling the progression of damage over time. We also incorporated a layer to enforce the previously defined domain constraints, ensuring that the predictions remain within realistic bounds. Despite extensive hyperparameter searches, these models encountered numerous setbacks and were unable to learn and predict effectively.

We then moved to other alternatives, given the promising results from the LSTM model we previously obtained in the domain constraint study, we decided to develop a model based on the Recurrent Neural Network (RNN) concept, but with Monodense layers. RNNs are well-suited for time series data due to their ability to capture temporal dependencies by maintaining a hidden state that is updated at each time step. This makes them an excellent choice for modeling sequential data where past information is crucial for making accurate predictions. Additionally, we incorporated the layer to enforce the previously defined domain constraints. Initially, all layers were set to be Monodense and the model was trained with a learning rate of 1×10^{-3} over 2000 epochs. Here are the results :

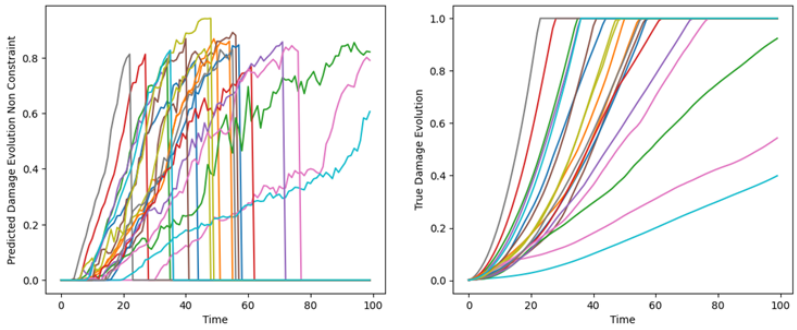

FIGURE 67 : RNN model with only Monodense layers

Remarks : While the model effectively adheres to the monotonicity constraints during the initial phases of damage evolution, there is a noticeable issue of saturation once $t_{\max}$ is reached. This saturation occurs when the model reaches the maximum damage point and fails to continue learning the damage progression effectively, indicating a limitation in handling long-term dependencies or extrapolation beyond the training data. From this model we can conclude that while the Monodense-constrained RNN model shows promise in enforcing monotonicity and providing smooth predictions, there is still this saturation issue.

Next, we tested the approach of implementing the Monodense layer only at the input layer to ensure the passage of the monotonicity indicators, while using standard dense layers for the remaining layers, the model was trained with a learning rate of 1×10^{-3} over 1000 epochs. Here are the results :

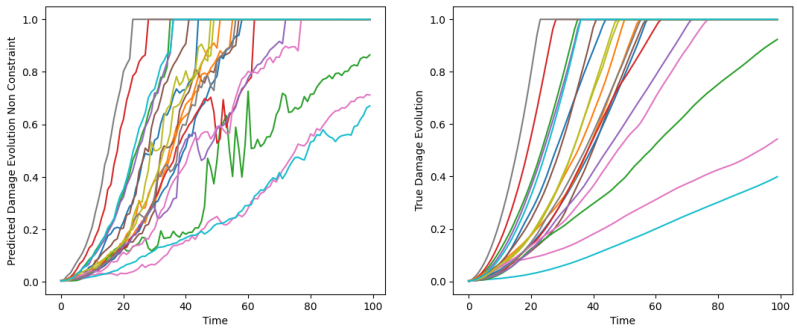

FIGURE 68 : RNN model with Monodense and dense layers

Remarks : Compared to the previous model where all layers were Monodense, this model shows an improved ability to capture the overall trend of damage evolution while maintaining monotonicity. The constrained model demonstrates smoother and more consistent trajectories, particularly in the early stages of the damage progression. This indicates that securing the passage of the monotonicity indicators at the input layer effectively guides the learning process, allowing the subsequent standard dense layers to refine the predictions without violating the monotonicity constraint.

However, we must admit that there are still some fluctuations observed in the mid to late stages of the predictions, and the predictions are not as smooth as we expected, these fluctuations could be due to the model's reduced ability to handle long-term dependencies with dense layers alone.

Based on the results obtained, we decided to conclude our study with Monodense layers and, in general, our exploration of monotonicity constraints. Despite the initial promise shown by integrating monotonicity indicators and constraints, the challenges encountered with learning stability and prediction accuracy led us to reevaluate our approach. Moving forward, other alternative methods should be considered, to potentially explore other architectures that can better capture the underlying data patterns and produce better predictions.

6.2.5 Analysis of the results

Domain Constraints

In general, we can conclude that the addition of domain constraints consistently improves the performance across all the models we tested, with the most notable improvements observed in the convolution and LSTM models. These results highlight the critical importance of incorporating domain-specific constraints to improve the accuracy of predictive models, especially in applications where adhering to physical or logical boundaries is essential like ours.

Monotonicity Constraints

The implementation of Monodense layers provided valuable insights into the effectiveness of the incoporation of monotonicity constraints in prediction tasks. Our findings highlight the importance of selecting appropriate layers for incorporating constraints and suggest that hybrid approaches, which combine Monodense layers with other architectures, may offer a more robust solution. Moving forward, exploring alternative architectures or approaches will be crucial for obtaining better results.

From this application case we can point out that whether is domain or monotonicity constraints they can significantly enhance the performance of predictive models. While domain constraints showed clear and consistent improvements across different models, the exploration of monotonicity constraints revealed the potential benefits and challenges of enforcing such properties.

The results suggest that a thoughtful integration of constraints, tailored to the specific needs of the task, can lead to more accurate and reliable predictions. Future research should continue to refine these approaches to further improve models.

7 Conclusions

7.1 Application case épures

Regarding the scalability of the approach where we included the constraints during training, it can be easily implemented. However, it may become laborious to order the products and spot the segments accurately. Despite this, the implementation of simple constraints has been proven effective, as the generated data respects the specified constraints. This demonstrates the potential of using constraints to guide the generation process, ensuring the compliance to contact constraints. Regarding the diversity study, as already discussed before the generations are in general good and further exploration could have been needed to get better results as we expected them.

7.2 Application case damage prediction

Incorporating constraints allows the model to focus on other important details during training, leading to improved overall performance. By adding constraints, we can employ simpler models to achieve the same task, which reduces computational complexity and enhances efficiency. Nevertheless, it remains crucial to continue exploring the implementation of monotonicity constraints, as

this could further refine the model's predictions and improve its robustness in capturing essential trends.

7.3 General Conclusions

We had the opportunity to test different types of constraints and we can conclude that some are easier to encode and implement than others. This variation highlights the need to carefully choose the appropriate constraints for a given task and the right type of models. Despite the advancements, a significant challenge persists in encoding domain knowledge into the form of constraints effectively. Ensuring that these constraints are both accurate and practical is essential for the model's performance. As shown the addition of hard constraints can greatly enhance the model's reliability and adherence to domain-specific requirements, although this may require further refinement.

Future work should focus on exploring and testing innovative approaches that integrate various types of constraints. This is a promising field that can significantly advance and expand the application of AI and deep learning models in fields where adherence to specific domain knowledge is essential.

8 Library

After concluding the exploration of the application cases, we created an internal Python library that encapsulates the primary domain approaches investigated during my internship. This library includes functionalities for imposing domain constraints, using MonoDense layers, and implementing the MultiplexNet architecture, among other features. Building this library was a great opportunity get experience on building a python library and to enhance the re-usability of my work.

9 Retrospective and feedback

My whole internship experience at Michelin was very enriching, I had the opportunity to experience what my future career will look like. I applied what I had learned during my studies in an industrial setting, enhancing my technical skills in Python programming and deep learning models. Additionally, I developed my management abilities, effectively organizing tasks and tracking progress. This experience also boosted my confidence in presentations.

It was a challenging experience for me at fist because it was my first experience in a job setting in a company, that was related to my studies, I was confronted to practices and behaviors which were not familiar to me, the meetings, the acronyms, the different departments and the small talk with colleagues. However, with the support of my supervisors and the data science team, I successfully adapted to the new environment

My time at Michelin allowed me to give that jump from from a very research oriented path I had lead until now to what I truly want for my future that is to work in a company.

I hope that the work I carried out will serve as a starting point for other developments and to the addition of domain knowledge into more projects at Michelin. Finally, it was overall a good experience and I am thankful for the opportunity I was given.

Références

[Cyb89] George Cybenko. "Approximation by superpositions of a sigmoidal function". In : *Mathematics of control, signals and systems* 2.4 (1989), p. 303-314.

[Gup+19] Akhil Gupta et al. "How to incorporate monotonicity in deep networks while preserving flexibility ?" In : *arXiv preprint arXiv :1909.10662* (2019).

[RPK19] M. Raissi, P. Perdikaris et G.E. Karniadakis. "Physics-informed neural networks : A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In : *Journal of Computational Physics* 378 (2019), p. 686-707. issn : 0021-9991. doi : https://doi.org/10.1016/j.jcp.2018.10.045. url : https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[Beu+21] Tom Beucler et al. "Enforcing analytic constraints in neural networks emulating physical systems". In : *Physical Review Letters* 126.9 (2021), p. 098302.

[Das+21] Tirtharaj Dash et al. "Using domain-knowledge to assist lead discovery in early-stage drug design". In : *International Conference on Inductive Logic Programming*. Springer. 2021, p. 78-94.

[Mon+21] João Monteiro et al. "Not too close and not too far : Enforcing monotonicity requires penalizing the right points". In : *eXplainable AI approaches for debugging and diagnosis*. 2021.

[Hoe+22] Nick Hoernle et al. "Multiplexnet : Towards fully satisfied logical constraints in neural networks". In : *Proceedings of the AAAI conference on artificial intelligence*. T. 36. 5. 2022, p. 5700-5709.

[NKW22] Niklas Nolte, Ouail Kitouni et Mike Williams. "Expressive Monotonic Neural Networks". In : *The Eleventh International Conference on Learning Representations*. 2022.

[RS23] Davor Runje et Sharath M Shankaranarayana. "Constrained monotonic neural networks". In : *International Conference on Machine Learning*. PMLR. 2023, p. 29338-29353.

[SP23] Alexander Scheinker et Reeju Pokharel. "Physics-constrained 3D convolutional neural networks for electrodynamics". In : *APL Machine Learning* 1.2 (2023).

[THH23] Jesus Tordesillas, Jonathan P How et Marco Hutter. "RAYEN : Imposition of Hard Convex Constraints on Neural Networks". In : *arXiv preprint arXiv :2307.08336* (2023).

[GDB24] Kshitij Goyal, Sebastijan Dumancic et Hendrik Blockeel. "DeepSaDe : Learning Neural Networks That Guarantee Domain Constraint Satisfaction". In : *Proceedings of the AAAI Conference on Artificial Intelligence*. T. 38. 11. 2024, p. 12199-12207.

[Zha+24] Jiachi Zhao et al. "Deep Isotonic Embedding Network : A flexible Monotonic Neural Network". In : *Neural Networks* 171 (2024), p. 457-465.

10 Annexes

10.1 State-of-the-art

10.1.1 Deep Isotonic Embedding Network : A flexible Monotonic Neural Network

In this paper [Zha+24] a new monotonic neural network named Deep Isotonic Embedding Network (DIEN) is developed, which uses different modules to deal with monotonic and non-monotonic features respectively, and then combines the outputs of these modules linearly to obtain the prediction result.

They introduce a new embedding tool called Isotonic Embedding Unit that is developed to process monotonic features and turn each one into an isotonic embedding vector. By converting non-monotonic features into a series of non-negative weight vectors and then combining them with isotonic embedding vectors that have special properties, they enable DIEN to guarantee monotonicity.

The introduction of a module called Monotonic Feature Learning Network to capture complex dependencies between monotonic features serves as a monotonic feedforward neural network with non-negative weights that can handle scenarios where there are few non-monotonic features or only monotonic features. In comparison to existing methods.

10.1.2 Enforcing Analytical Constraints in Neural Networks Emulating Physical Systems

Neural networks can emulate nonlinear physical systems with high accuracy, yet they may produce results that are physically-inconsistent when violating fundamental constraints. In this paper [Beu+21] a systematic way of enforcing nonlinear analytic constraints in neural networks via constraints in the architecture or the loss function is proposed to enforce conservation laws to within machine precision without degrading the performance of the model. The main field of application is climate modeling,

Different types of architectures such as **ACNets** architecture constrained networks, **UCNets** unconstrained networks and **LCNets** loss constrained networks are presented. When considering a generic regression task subject to analytical constraints that are generally written using analytical functions of the system's variables and may not be linear, so, a framework to write them as a linear system is proposed. More precisely when implementing the **ACnet** architecture outputs are calculated using a standard neural network while the remaining outputs are calculated as residuals from the fixed constraints layers as shown in figure 69.

Some advantages of this approach are the fact that constraints are directly applied to the architecture of the NN. ACnets are applicable to a broad range of constrained optimization problems. While the definition of the constraints might be difficult depending of the complexity of the non linear constraints and LCNets might not fully impose the respect of the constraints.

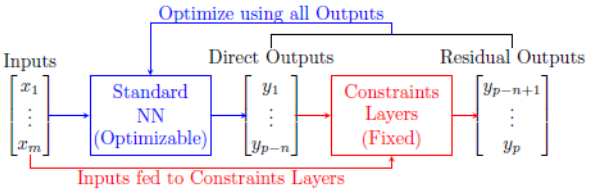

FIGURE 69 : ACnet Architecture

10.1.3 RAYEN : Imposition of Hard Convex Constraints on Neural Networks

Rayen [THH23] is introduced as a framework to impose hard convex constraints on the output or latent variable of a neural network. It guarantees that for any output or any weights of the Neural network, the constraints are satisfied at all times, it does not rely on soft constraints. RAYEN supports any combination of linear, convex quadratic, second-order cone (SOC), and linear matrix inequality (LMI) constraints, achieving a very small computational overhead compared to unconstrained networks.

Convex constraints are widely used in various areas of engineering. Recently, there has been an extensive use of neural networks in these applications due to their expressive power. However, the lack of constraint satisfaction guarantees greatly limits their applicability.

10.1.4 DeepSaDe : Learning Neural Networks that Guarantee Domain Constraint Satisfaction

An approach to train neural networks so that they can enforce a wide variety of constraints and guarantee that the constraint is satisfied by all possible predictions, they focus on semantic domain constraints.
This approach [GDB24] is build on the idea where learning linear models are formulated as a constraint satisfaction problem (CSP). Then to make this idea applicable to neural networks, the two key modifications to the network's architecture and training procedure are :

- Propagating the constraints over the network layers to the last layer, which involves adding skip connections that copy the input to the penultimate layer and deriving bounds on the penultimate layer from the bounds on the input
- Training the network using a hybrid procedure that uses MaxSMT to determine the weights in the output layer and gradient descent for all other

MaxSMT

MaxSMT stands for Maximum Satisfiability Modulo Theories. It is an extension of SAT solving that can take background theories into account and that distinguishes soft and hard constraints : it returns a solution that satisfies all hard constraints and as many soft constraints as possible.
This approach is flexible enough to enforce a wide variety of domain constraints and is able to guarantee them in neural network.