Neural learning and validation of hierarchical multi-criteria decision aiding models with interacting criteria

Roman Bresson

université
PARIS-SACLAY

Neural learning and validation of hierarchical multi-criteria decision aiding models with interacting criteria

Apprentissage neuronal et validation de modèles hiérarchiques d'aide à la décision multicritère présentant de l'interaction entre les critères

Thèse de doctorat de l'Université Paris-Saclay

École doctorale : n°580 : sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat: Informatique
Graduate School : Informatique et sciences du numérique, Référent : Faculté des sciences d'Orsay

Thèse préparée dans l'unité de recherche: Laboratoire interdisciplinaire des sciences du numérique, (Université Paris-Saclay, CNRS), sous la direction de Johanne COHEN, Professeur, le co-encadrement de Christophe LABREUCHE, Docteur et Ingénieur de Recherche (Thales Recherche et Technologie).

Thèse soutenue à Paris-Saclay, le 02 Février 2022, par

Roman BRESSON

Composition du jury

Patrice PERNY Professeur, Sorbonne Université, Paris 6	Rapporteur & Président
Andrea PASSERINI Professeur Associé, Université de Trente, Italie	Rapporteur & Examinateur
Hendrik BLOCKEEL Professeur, KU Leuven, Belgique	Examinateur
Krzysztof DEMBCZYŃSKI École Polytechnique de Poznań, Pologne	Examinateur
Eyke HÜLLERMEIER Professeur, Université Louis-et-Maximilien, Munich, Allemagne	Examinateur
Michèle SEBAG Professeur, Université Paris-Saclay	Examinatrice
Johanne COHEN Professeur, Université Paris-Saclay	Directrice

THESE DE DOCTORAT
NNT : 2022UPASG008

université PARIS-SACLAY | ÉCOLE DOCTORALE Sciences et technologies de l'information et de la communication (STIC)

Titre: Apprentissage neuronal et validation de modèles hiérarchiques d'aide à la décision multicritère présentant de l'interaction entre les critères

Mots clés: Apprentissage Automatique, Aide à la Décision Multicritère, Intégrale de Choquet, IA de Confiance

Résumé: L'aide à la décision multicritères (ADMC) est un domaine qui vise à aider des décideurs experts (DE) pour des problèmes tels que la sélection, le classement ou la classification d'alternatives définies par plusieurs attributs qui peuvent intéragir. Ces modèles ne sont pas ceux qui prennent pas la décision, mais ils apportent une assistance au DE lors du processus. Il est donc crucial que le modèle offre au DE des moyens d'interpréter ses résultats. Ceci est en particulier vrai dans des contextes critiques où les erreurs peuvent avoir des conséquences désastreuses. Il est par conséquent indispensable que les modèles d'ADMC soient intelligibles, interprétables et que leur comportement soit fortement contraint par des connaissances provenant d'une expertise dans le domaine. De tels modèles sont généralement construits par une interaction (questions/réponses) avec un DE, par le biais de méthodes issues de la recherche opérationnelle.

D'autre part, l'apprentissage automatique (ML) fonde son approche sur l'apprentissage du modèle optimal à partir de données d'ajustement. Ce domaine se concentre généralement sur les performances du modèle, en adaptant les paramètres de modèles complexes (dits boîtes noires) pour obtenir une erreur statistiquement faible sur de nouveaux exemples. Bien que cette approche soit adaptée à de nombreux contextes, l'utilisation de modèles boîtes noires est inconcevable dans les cas usuels d'ADMC, car ils ne sont ni interprétable, ni facilement contraignables.

Cette thèse fait le pont entre ces deux domaines. Nous nous concentrons sur une certaine classe de modèles d'ADMC, appelés intégrales de Choquet hiérarchiques utilitaires (ICHU). Notre première contribution, qui est théorique, est de montrer l'identifiabilité (ou l'unicité de la paramétrisation) des ICHUs. Ce résultat motive notre seconde contribution : le framework NEUR-HCI, une architecture de modules de réseaux de neurones qui peuvent apprendre les paramètres d'un ICHU. En particulier, tous les modèles NEUR-HCI sont garantis comme étant formellement valides, répondant aux contraintes qui conviennent à de tels modèles (monotonie, normalisation), et restent interprétables.

Nous montrons empiriquement que les modèles NEUR-HCI sont performants sur des ensembles de données artificielles et réelles, et qu'ils présentent une stabilité remarquable, ce qui en fait des outils pertinents pour alléger l'effort d'élicitation de modèles lorsque les données sont facilement disponibles, et permet leur utilisation comme outils d'analyse appropriés pour identifier certains phénomènes sous-jacents dans les données.

Title: Neural learning and validation of hierarchical multi-criteria decision aiding models with interacting criteria

Keywords: Machine Learning, Multi-Criteria Decision Aiding, Choquet Integral, Trustable AI

Abstract: Multicriteria Decision Aiding (MCDA) is a field that aims at assisting expert decision makers (DM) in problems such as selecting, ranking, or classifying alternatives defined on several interacting attributes. Such models do not make the decision, but assist the DM, who takes the final decision. It is thus crucial for the model to offer ways for the DM to maintain operational awareness, in particular in safety-critical contexts where errors can have dire consequences. It is thus a prerequisite of MCDA models to be intelligible, interpretable, and to have a behaviour that is highly constrained by information stemming from in domain knowledge. Such models are usually built hand in hand with a field expert, obtaining information through a Q&A procedure, and eliciting the model through methods rooted in operations research.

On the other hand, Machine Learning (ML), and more precisely Preference Learning (PL), bases its approach on learning the optimal model from fitting data. This field usually focuses on model performances, tuning complex black-boxes to obtain a statistically low error on new examples cases. While this is adapted to many settings, it is out of the question for decision aiding settings, as neither constrainedness nor intelligibility are available.

This thesis bridges both fields. We focus on a certain class of MCDA models, called utilitaristic hierarchical Choquet integrals (UHCI). Our first contribution, which is theoretical, is to show the identifiability (or unicity of the parameterization) of UHCIs This result motivates our second contribution: the NEUR-HCI framework, an architecture of neural network modules which can learn the parameters of a UHCI. In particular, all NEUR-HCI models are guaranteed to be formally valid, fitting the constraints that befit such a model, and remain interpretable.

We show empirically that NEUR-HCI models perform well on both artificial and real dataset, and that they exhibit remarkable stability, making it a relevant tool for alleviating the model elicitation effort when data is readily available, along with making it a suitable analysis tool for indentifying patterns in the data.

Maison du doctorat de Université Paris-Saclay
2e étage, aile ouest, École normale supérieure Paris-Saclay
4 avenue des Sciencs
91190 Gif-sur-Yvette, France

CONTENTS

Part I

Introduction

CHAPTER 1

INTRODUCTION

1.1 Context and Motivation

Preference and scoring models are pervasive in our world. They take on many forms and find applications as diverse as recommending a movie to a streaming plaftorm's users, suggesting a treatment for a medical doctor's patient, or designating a suspect vehicle to be intercepted by the authorities.

Machine Learning (ML) is a set of methods for building models whose parameters are extracted from data through statistical optimization methods. In particular, Preference Learning is the subfield which aims at building and parameterizing preference models from data. Preference Learning methods are thus expected to yield accurate predictive models, learning from observations which are, by nature noisy and imperfect. Such models are expected to determine a preferred *alternative* among a set of several ones.

Nonetheless, ML is, by nature, highly statistical, and the trained models usually do not offer the trustability needed in *safety-critical* contexts. Safety-critical fields, like medicine, defense, or air-traffic management, are characterized by the fact that a single error can have devastating consequences, from system failures to loss of lives. In these cases, where much is at stake, it is not enough for a model to simply perform well statistically. It is then often necessary that the used models behave in a specific way; there might thus be some strong constraints, extracted from domain-specific knowledge, that the model must fit in order to be accepted by a field-expert, and thus used in practice.

Moreover, if the model is to be trusted by such an expert, meeting these constraints is necessary, but not sufficient. Indeed, the user might want to be able to appreciate the (context-dependent) relevance of a suggested alternative, or a

given score; in order to make the final decision themselves, thus using the model as a decision-aiding tool rather than a decision-making one. On another side, users might want to verify the model, after training and before using it, checking if it fits their high-level idea of what its behaviour should be. For both of these reasons, it is necessary that the model be interpretable.

A field which adresses such concerns is *Multi-Criteria Decision Aiding* (MCDA). MCDA focuses on building, with the help of an expert, preference and decision models based that are highly constrained, interpretable, and trustable, thus clearly adapted for safety-critical contexts. In this thesis, we focus on models based on an aggregation function called the *Choquet integral* (CI), which belongs to a framework called *Multi-Attribute Utility Theory* (MAUT). MAUT models aim at assigning, to any given alternative, a score, or *global utility*, which corresponds to its attractiveness. One can easily see how such scores can then be used as bases for ranking, pairwise preferences, or for sorting the alternatives into diverse preference classes.

We chose to work with the CI, as it offers a good trade-off between representation power (being a non-linear aggregator able to model several types of interactions) and interpretability (its complexity being limited from the selected CI parametric class). Moreover, it has the advantage of filling by-design many desireable properties, such as being monotonic and compensatory w.r.t. its inputs, regardless of its parameterization. Finally, it is a well known model, which has been used in decision aiding for decades, and is starting to gain interest in ML.

Still, when considering up to some dozen criteria, models can become hard to both design and interpret. Hierarchical models, and in particular *hierarchical Choquet integrals* (HCIs) address this limitation through a divide-and-conquer approach, gradually aggregating the original criteria to form higher-level (abstract) criteria. As a small number of criteria are aggregated at each step, the model recommendations can easily be traced, verified and understood. Moreover, as each individual CI is compensatory ad monotonic w.r.t. its inputs, we have the guarantees that the HCI has the same properties.

It is also common in MAUT to use *marginal utility functions* for rescaling and normalizing the attributes. Our approach is thus adapted to learning such functions, which associate to each attribute a satisfaction depending on its value, independently from the values of the other attributes. These marginal utilities take values in the interval $[0, 1]$. In order to retain interpretability, we focus on monotonic or bitonic marginal utilities, which are usual in MCDA. Such rescalings allow to increase greatly the representation power of the model class, while retaining a high interpretability. Below, we write UCI (resp. UHCI) a model composed of a CI (resp. HCI) fitted with marginal utilities.

Traditionally, in decision-aiding, UHCIs are manually designed by domain ex-

perts. This means that the latter must be able to define:

- a hierarchy (i.e. a tree of successive aggregations of the criteria)
- the marginal utilities
- the parameters (weights) of the aggregators

Usually, those elicitation methods rely on approaches from *operations research* (OR): the domain knowledge is represented as constraints; combinatorial optimization methods can then be used to optimize each aggregator and find a proper model parameterization thereof. The manual design methodology however faces two well-known bottlenecks: the shortage of the expert time, and the law of diminishing returns, making it increasingly more difficult to improve the models as their quality increases.

Moreover, these OR-based approaches require to elicit each part (aggregators and marginal utilities) of the model separately. In many situations, the *decision maker* (DM) cannot effectively provide local preferences without provoking global inconsistencies (namely, the DM might disagree with the consequence at holistic level of all preferential information they provided at local level).

In contexts where user preferences/constraints are available from data, an alternative to manual model design is offered by automatically building preference models through supervised machine learning (tackling classification, regression and/or preference learning problems depending on the available data); nonetheless, these approaches are currently limited to single CIs rather than hierarchical ones.

In order to exploit the best of both worlds (MCDA and ML), we propose in this thesis a framework for learning UHCIs, from data. As a consequence, we can both exploit noisy and large datasets for building a model, but can still guarantee and validate the model, exploiting decades of formal work on MAUT models, which show their interpretability, constrainedness, and all in all, trustability. Thus, the learned models are thus exploitable in safety-critical contexts, once validated by a field expert.

1.2 Main Contributions

Our main theoretical contribution of this thesis is the proof of identifiability of the class of UHCI functions. That is, under mild assumptions, a given UHCI can have a single hierarchy, a single set of marginal utilities, and a single set of weights (parameters of each aggregator). This unicity is highly interesting, as it helps building trust in a model learned from otherwise statistical data.

The main technical contribution of this thesis is the NEUR-HCI framework, which automatically extracts the parameters of a UHCI model, along with its marginal utilities, from: i/ the available data, ii/ the hierarchical aggregation tree-structure defined on the criteria.

The model verification and interpretation are ensured as NEUR-HCI enforces *by design* all HCI model constraints (e.g. monotonicity and idempotency). Specifically, NEUR-HCI automatically translates the HCI tree structure into the architecture of a neural net, the weights of which are optimized through back-propagation from the available data. NEUR-HCI thus gets the best of both worlds, retaining the interpretability of the HCI representation class, and the affordable training complexity of neural nets.

Interestingly, NEUR-HCI also learns the marginal utility functions, supporting a simpler description of the model. As said, a NEUR-HCI model is modular by design; this enables experts to validate, modify, or re-structure the tree whenever they need (in which case it might be necessary to re-train the model, or at least parts of it).

1.3 Publications

This PhD yielded four publications:

Neural Representation and Learning of Hierarchical 2-additive Choquet Integrals
R Bresson, J Cohen, E Hüllermeier, C Labreuche, M Sebag; International Joint Conference on Artificial Intelligence (IJCAI-20), Pages 1984-1991

Learning 2-additive Hierarchical Choquet Integrals with non-monotonic utilities
R Bresson, J Cohen, E Hüllermeier, C Labreuche, M Sebag; Proceedings, From Multiple Criteria Decision Aid to Preference Learning, (DA2PL2020)

Evaluating the stability of the Neur-HCI framework
R Bresson, J Cohen, E Hüllermeier, C Labreuche, M Sebag; Actes de la conférence (CAID 2020), Pages 120-128

On the Identifiability of Hierarchical Decision Models
R Bresson, J Cohen, E Hüllermeier, C Labreuche, M Sebag; 18th International Conference on Principles of Knowledge Representation and Reasoning (KR2021), Pages 151-162

1.4 Organization of this Manuscript

This thesis is organized as follows:

1.4.1 Part II: Background and Existing Work

In this part, we present the basic notions necessary for establishing the context of our work. This part is divided into two chapters, as this thesis is at the crossroad of two large fields. First, in Chapter 2, we introduce the field of Multi-Criteria Decision Aid (MCDA).

We first present the basic notions and motivations of the field; in particular, the notion of alternatives defined over attributes. We detail the motivations of the field; that is, to build criteria that will represent how satisfying an alternative is on a given attribute, in order to establish decision models that represent a decision maker's preferences. Such models are interpretable, and highly constrained (trustable), making them suitable for use in safety-critical systems. We detail several classes of models, along with model-building methods.

We focus in particular on decomposable models. Such models have the following interesting property: the satisfaction brought by a given attribute does not depend on the value of the other attributes. While this restricts the possible behaviours of the model, it also brings intelligibility to the model, which can then be interpreted much more easily.

The specific family of models on which we focus is the Choquet Integral (CI), a generalization of the weighted sum which allows to take interactions among criteria (synergy, independance and redundancy) into account. We generalize to its hierarchical version (HCI). The HCI is a highly interpretable aggregator, which will be used to compute a global score, for an alternative, given its attribute-wise satisfactions. This requires to have the latter readily available, which is not always the case when we are given the raw values on the attributes. As a consequence, an HCI can also be combined with so-called *marginal utilities*, which are 1-dimensional real functions on the attributes which allow to compute the satisfaction brought by a certain value of the given attribute. An HCI thus fitted with marginal utilities is called a UHCI. We detail these models, as they are at the center of this thesis' contributions.

UHCIs have the advantages of MCDA models, in that it is interpretable and constrained. Nonetheless, it also suffers the usual drawbacks of this field, which is strongly linked to operations-research. It requires consistent preferential information, obtained often iteratively through an interaction with a field expert. This process is thus costly, and its efficiency is bottlenecked by the human source of information. Finally, it does not allow to exploit existing and noisy data, which

might be readily available in certain contexts.

As a consequence, we turn our sights to Machine Learning (ML). In Chapter 3, we introduce the domain of supervised ML. We present the underlying concepts and widely used methods. We then give a brief overview of a family of models called Neural Networks (NN). Indeed, such models have an inherent hierarchical structure, which is perfectly adapted for representing UHCIs.

We present recurrent issues present in ML, in particular those due to uncertainy, or problematic behaviours, such as overfitting.

We then introduce basic notions of Preference Learning, a domain which focuses on learning preference models from data, as a more statistical and data-driven approach to the decision problems that MCDA tackles in a constraint-driven way.

We develop in particular ML methods aimed at learning models from the field of MCDA, but conclude that, while the CI has been a point of interest for some years now, UHCIs have never been learned before.

1.4.2 Part III: Theoretical Contribution

Part III is the first of the two parts dedicated to this thesis's contributions. It consists of a proof of identifiability of the UHCI model; that is, we show that there is only a single parameterization possible for a given model. This means that, for a given UHCI $\mathcal{F}$ which respects certain mild conditions, then $\mathcal{F}$ can only have:

- a single hierarchy;
- a single set of marginal utilities;
- a single set of weights for its aggregators.

This is a very interesting result for obtaining a trustable model. Indeed, a UHCI's intellegibility and interpretation lies in its parameters. Ensuring the existence of a single parameterization means ensuring that there cannot be two or more contradicting interpretations on a given model; a fact which would render the model effectively untrustable.

This proof is made in two steps, split among both chapters of the part. First, Chapter 4 presents the proof of unicity when the model is defined on a fixed hierarchy. That is, we suppose a given hierarchy, and show that two models that are equal everywhere on the attribute space have necessarily the same aggregators and marginal utilities. We proceed mainly by exploring the vertices of the input hypercube, which allows us to single-out each element and show its unicity.

Then, Chapter 5 generalizes to the case where the hierarchy is free. We proceed by considering another representation of the UHCI. Indeed, this class of models, given piecewise-C^1 marginal utilities (a weak assumption), is itself piecewise-C^1.

Our strategy is thus to study the borders between each region on which a UHCI is C^1. In doing so, we are able to show that under two mild assumptions, these borders are unique for a given model, and only a single model can have those borders, thus effectively completing the proof.

An added interest in this proof is that it is an encouraging result for the learning of UHCIs using statistical methods.

1.4.3 Part IV: Technical Contribution

In this part, we present the main technical contribution of this thesis. This contribution is the NEUR-HCI framework, which is a type of specific neural modules (i.e. small neural networks), each representing a specific part (CI aggregator, or marginal utility) of an UHCI model.

Chapter 6 presents modules implementing marginal utilities. We treat 4 types of marginal utilities:

- non-decreasing;
- non-increasing;
- single-peaked (non-decreasing then non-increasing);
- single-valleyed (non-increasing then non-decreasing).

These modules are built so that they can represent any (and only) marginal utility of their respective type. Their design guarantees that any constraint that are required by the model, in terms of monotonicity and normalization, be met, so that the model remains valid at all time.

We also present a so-called *selector* module, which allows to chose, among the four types presented above, the one most adapted to the training data.

Then, Chapter 7 introduces the neural architectures that represent CI-based aggregator functions. We present three types of Choquet integrals:

- 2-additive (which allows interactions of up to two criteria at a time)
- a subset of the 3-additive Choquet integrals (which allows interactions of up to 3 criteria at a time)
- general (any Choquet integral of the given dimension)

Just like for the marginal utility modules, we ensure by design the validity of the CI represented by the module, in terms of formal constraints.

Finally, Chapter 8 explains how we train a neural network built from the modules introduced in the two previous chapters. The training settings are, namely:

- **Regression:** the model is given a set of alternatives, and their expected scores, and learns to predict scores for new alternatives
- **Classification:** the model is given a set of alternatives, and their preference class (very bad, bad,..., very good), and learns to classify new alternatives
- **Pairwise preference learning:** the model is given pairs of alternatives, along with the information of which of both alternatives is prefered to the other. It then learns to predict new preferences

We also have present a theorem of validity, showing that any NEUR-HCI network is a valid UHCI, and that any UHCI with the same types of marginal utilities and additivity constraints as our modules can be represented by an NEUR-HCI network.

It is to be noted that this approach was validated on an internal Thales application.

1.4.4 Part V: Experimental Results

In this section, we test the performance, robustness and stability of the models presented in part IV. We do so on both artificially generated and real data. We show that not only can NEUR-HCI models learn efficiently a model that fits the data well, they also prove stable; that is, given enough data, models trained on similar data will have similar parameters and interpretation. This is highly relevant for trusting the model, and thus for applications in safety-critical applications.

1.4.5 Part VI: Perspectives and Conclusions

We finally present perspectives of future work, which could be done to iterate on the work presented in this thesis. These include extensions of the learned models, along with possible theoretical work to develop new indicators for interpreting MCDA models.

Part II

Background and Existing Work

CHAPTER 2

MULTI-CRITERIA DECISION AIDING

Contents

2.1 Introduction

Decision Aiding (DA) was defined by B. Roy in 1996 as "the activity of the person who, through the use of explicit but not necessarily completely formalized models, helps obtain elements of responses to the questions posed by a stakeholder of a decision process. These elements work towards clarifying the decision and usually towards recommending, or simply favouring, behaviour that will increase the consistency between the evolution of the process and this stakeholder's objectives and value system" [Roy, 1996].

In many application domains related to decision making and artificial intelligence at large, an essential requirement is to gain the users' trust [O'Neill, 2016]. To this end, the model must be *interpretable*, that is, a decision maker must understand which criteria influence the decision, how, and to which extent; in other words, they must be able to trace back the assessment of an alternative to the criteria involved. In some cases, syntactic constraints (e.g. monotonicity) might be enforced to facilitate the interpretation of the model; in other cases, specific domain knowledge is available (e.g. implying some preferences w.r.t. some criteria, everything else being equal), and the model must comply with this prior knowledge. Naturally, the trust-worthiness of the model is all the more important in safety-critical contexts. We now detail how such preference models are established in the context of multi-attribute utility theory.

We focus in this chapter on a subfield of DA, Multi-Criteria Decision Aid (MCDA). This field aims at building and analyzing models for representing the preferences of a decision maker (DM) over *alternatives* defined by their values w.r.t. several, possibly conflicting *criteria*. We also introduce the models on which this thesis will focus.

2.2 Notations and Definitions

2.2.1 Alternatives and Attributes

In the usual language, *alternatives* are the options given as possible choices in a decision problem. The decision maker tries to select the "best" option among the set of given alternatives, a notion which is further formalized in MCDA.

In MCDA, an alternative x is represented as a vector of its performance measured according to various attributes. Formally, let us write n the number of attributes used to evaluate our alternative, and $N = \{1, ..., n\}$ the set of the indices of the attributes. We then have n sets $X_1, ..., X_n$, respectively containing all the possible values for attributes 1 to n. By abuse of notations, we assimilate an attribute to its index, it will thus be common to see "attribute i" instead of "the ith attribute".

We also write $X = X_1 \times ... \times X_n$ the cartesian product of all of those sets; and X is the set of alternatives. An alternative is thus a point written in the vector space spanned by the basis composed of all of the X_i sets.

Example 1. *A decision maker (DM) wishes to buy a house. Let us assume that houses are defined on seven attributes, each defined on its own unit/set of values:*

1. *House surface area (sq. meters)*
2. *Garden surface area (sq. meters)*
3. *Garage (yes/no)*
4. *Distance to a large road (km)*
5. *Distance to public transportation (km)*
6. *Distance to downtown (km)*
7. *Price (€)*

House	Surface m^2	Garden m^2	Garage (yes/no)	Road km	Transp. km	Downtown km	Price €
h_1	50	100	No	0.1	0.	0.	400,000
h_2	110	150	Yes	0.5	3.	4.	500,000
h_3	150	0	No	1.	0.5	0.5	450,000
h_4	150	30	No	0.1	5.	3.	300,000
h_5	500	1000	Yes	5.	5.	10.	1,500,000

Table 2.1: Alternatives h_1 to h_5 for the pairwise learning setting

They have a set of 5 *real estate ads (the alternatives) to chose from, written* h_1 *to* h_5*, given in table 2.1. As we see, they are described by their values on each attribute.*

In this example, we have:

- $X_1 = X_2 = X_4 = X_5 = X_6 = X_7 = \mathbb{R}_+$
- $X_3 = \{$*Yes, No*$\}$

2.2.2 Formalizing preferences

2.2.2.1 Definitions

We present here a mathematical formalization of the notion of preference, which will be used throughout this thesis.

Definition 1. *Let a relation* $P \subseteq (X \times X)$*, called strict preference, and a relation* $I \subseteq (X \times X)$*, called indifference on the element of* X*.*

We assume the following properties on P *and* I*:*

- I ***is symmetric:*** $xIy \Rightarrow yIx$
- I ***is reflexive:*** $\forall x \in X : \ xIx$
- P ***is asymmetric:*** $xPy \Rightarrow \neg yPx$
- $P \cup I \cup P^{-1}$ ***is complete:*** *for* $x, y \in X$*, either* xPy*,* yPx *or* xIy

Then $R = P \cup I$ *is called a* ***complete preference relation*** *on* X*.*

We focus in particular on the widely used *weak orders* (also called complete preorders), which are a sub-class of preference relations:

Definition 2. *A* ***weak order*** *is a preference relation* $R = P \cup I$ *such that:*

- *I is transitive: $[xIy$ and $yIz] \Rightarrow xIz$*
- *P is transitive*
- *R is reflexive and complete*

In particular, R allows elements of X to be equivalent, even though they are different.

From now on, we write P as $\succ$, I as $\sim$, and R as $\succeq$. We use the following terminology:

- $x \sim y$ means that x is *equivalent* to y
- $x \succ y$ means that x is (strictly) *preferred to*, or better than, y
- $x \succeq y$ means that x is *at least as good as* y, i.e. $x \sim y$ or $x \succ y$

2.2.2.2 Weak Separability

In the remainder of this thesis, we use the following compound notation:

Let $i \in N,\ \mathbf{a} \in X,\ x_i \in X_i,\ \mathbf{b} = (x_i, \mathbf{a}_{-i})$ is the alternative such that:

$$b_k = \begin{cases} x_i & \text{if } k = i \\ a_k & \text{otherwise} \end{cases} \tag{2.1}$$

We also write X_{-i} the subspace spanned by all attributes, except i. Weak separability [Krantz et al., 1971, Bouyssou et al., 2006] is the property on a preference relation $\succeq$ that states that:

$$\begin{gathered} \forall i \in N,\ \forall a_i, b_i \in X_i,\ \forall x_{-i}, y_{-i} \in X_{-i} : \\ (a_i, x_{-i}) \succeq (b_i, x_{-i}) \iff (a_i, y_{-i}) \succeq (a_i, y_{-i}) \end{gathered} \tag{2.2}$$

This means that the preference relation on attribute i is independent of the value on any other attribute.

Example 2. ***Ex. 1 continued.*** *The DM prefers a larger surface area. If their preference relation satisfies weak independence, this means that, for two houses that have the same values on all attributes except the surface area, then the larger*

house will be preferred.

On the other hand, assume a decision problem where a doctor should evaluate the healthiness of a patient, given several attributes, two of which would be their average resting heart rate and their age. Nonetheless, we know that the ideal heart rate is highly dependant on the age:

Age	Average heart rate (bpm)
0.25 years	143
50 years	75

Table 2.2: Optimal heart rate for some ages

Assume two subjects, a_1 *of age* 0.25 *years (*3 *months), and* a_2 *of age* 50*. We write these alternative as vectors (age, heart rate).*

Then, we have $(0.25, 143) \succ (0.25, 75)$*, as a* 3 *months old baby with only* 75 *bpm has severe bradychardia (insufficient heart rate), but has a normal heart rate with* 143 *bpm.*

On the other hand, we have $(50, 143) \prec (50, 75)$*, as a* 50 *year old person with* 143 *bpm has severe tachychardia (too high a heart frequency), but has a normal heart rate with* 75 *bpm.*

In this case, weak separability does not hold, as the satisfaction brought by the heart rate is highly dependent on the age of the subject.

2.2.2.3 Marginal Preferences and Criteria

Weak separability on $\succeq$ allows us to introduce a marginal preference relation $\succeq_i$ among the elements of X_i, with $\succ_i$ its asymmetric part and $\sim_i$ its symmetric part.

Example 3. ***Ex. 1 continued****. A DM finds* €$400,000$ *to be a more satisfying price (attribute 7) than* €$1,500,000$*. We can thus write:*

$$400,000 \succ_7 1,500,000$$

On the other hand, they find $500m^2$ *to be a more satisfying surface area (criterion 1.) than* $100m^2$*. Thus:*

$$500 \succ_1 100$$

Finally, the DM is indifferent about whether garden sizes of $300m^2$ *and* $400m^2$ *(attribute* 2.*), as both are wide enough not to have any space issue. Thus, we write:*

$$300 \sim_2 400$$

An attribute, together with a total preference relation on its values, is called a *criterion*. By abuse of notation, we also use N as our family of criteria. This family must meet three main properties, defined in [Roy, 1999]:

- **Exhaustiveness**: if two alternatives have similar performance on all criteria, the DM should not be able to say that one is preferred to the other (the family can illustrate all of the preference information).
- **Cohesiveness**: if two alternatives **a** and **b** have the same performance on all of the criteria except on criterion i, and $a_i > b_i$, then the DM recognizes that **a** is preferred to **b**.
- **Nonredundancy**: if one of the criteria is removed from the family, at least one of the properties above stops holding.

2.2.3 MCDA problems

The aim of MCDA is to build preference models which reflect the DM's preferences on the space of alternatives. Such models are then to be used for assisting the decision maker in its later choices. According to [Danila, 1986] and [Roy, 1996], the three following decision problems are often met by DMs: **Choice, Ranking** and **Sorting**.

Choice: A **Selection** or **Choice** problem is when the DM must chose the best alternative among a finite set of possibilities.

Example 4. ***Example 1 cont'd***
The DM considers that house 3 *is their favorite of the* 5 *given in Table 2.1.*

Ranking: A **Ranking** problem is when the DM wants to order alternatives from a set according to their preference; that is, from best to worst, with a preference ordering on the set of alternatives.

Example 5. ***Example 1 cont'd***
The DM gets the following order from his preferences:

$$h_3 \succ h_1 \succ h_5 \succ h_2 \sim h_4$$

with $\succ$ *denoting preference, and* $\sim$ *denoting equivalence.*

Sorting: A **Sorting** problem is when the DM wants to assign an alternative to a preference class. Such classes are so there is a total order between all classes, from most to least preferred.

Example 6. ***Example 1 continued***
The DM sorts the houses among 3 *preference classes:*

- *Satisfying:* h_3
- *Average:* h_1 *and* h_5
- *Not-satisfying:* h_2 *and* h_4

2.3 Multi-Attribute Utility theory

The focus of this thesis is on scoring functions, also called utility functions. *Utility functions*, also simply called *utilities*, stem from economics, and are real functions that aim at representing the satisfaction provided by an alternative to the DM. Multi-Attribute Utility Theory (MAUT) [Fishburn, 1970, Abdellaoui and Gonzales, 2013] generalizes this notion to alternatives defined on several criteria, such as those found in MCDA.

2.3.1 Global Utility

A *utility function* $\mathcal{F} : X \to \mathbb{R}$ is a function that is increasing with the satisfaction of the DM. Formally, $\mathcal{F}$ is a suitable utility function for representing the DM's preferences if and only if

$$\forall a, b \in X^2, a \succeq b \iff \mathcal{F}(a) \geq \mathcal{F}(b)$$

Example 7. ***Example 5 continued:*** *a utility function that satisfies the preference relations given among the houses would satisfy:*

$$\mathcal{F}(h_3) > \mathcal{F}(h_1) > \mathcal{F}(h_5) > \mathcal{F}(h_2) = \mathcal{F}(h_4)$$

2.3.1.1 Marginal Utilities

Utility functions can also be defined on single criteria and are then called *marginal* or *local* [Keeney et al., 1979] (as opposed to *global* when they compare complete alternatives). In the same way as the global utilities, marginal utilities are built to represent the satisfaction of a decision maker w.r.t. the value on a single criterion. Formally, let $u_i : X_i \to \mathbb{R}$, let $a, b \in X^2$, $u_i(a_i) \geq u_i(b_i)$ means that a is more satisfying on criterion i than b is (i.e. $a_i \succeq b_i$).

Example 8. *Given the preferences given in Example 3, we would have:*

$$u_7(400,000) \geq u_7(1,500,000)$$
$$u_1(500) \geq u_1(100)$$
$$u_2(300) = u_2(400)$$

More generally, as the DM always prefers a lower price to a higher one (the cheaper the better), we would have a non-increasing marginal utility u_1; on the other hand, the bigger the surface area, the better, thus, u_2 would be non-decreasing.

Usually, in MCDA, the attributes are built with the DM in order to ensure that the marginal utilities be monotonic w.r.t. the value on the criterion (bigger is better or smaller is better). This is possible as MCDA models are built hand in hand with the DM, and it allows the local utilities to simply be re-scaling functions. A common extension is the use of single-peaked or single-valleyed functions. A single-peaked function is a function which is non-decreasing up to a maximal value, then non-increasing; a single-valleyed one is the opposite: non-increasing then non-decreasing.

Example 9. ***Example 1 continued:*** *Consider attribute 4. distance to a large road (in kilometers). The DM prefers*

- *to be close enough for easy access*
- *to be far enough that they are not bothered by the noise*

Assuming 0.1km is too close, 2km is optimal, and 10km is too far. This preference is reflected by $u_4(2) > u_4(0.1)$ and $u_4(10) < u_4(2)$.

The adapted marginal utility function is thus single-peaked, with a peak (or plateau) in 2.

2.3.1.2 Commensurability

It is a suitable property to have commensurable marginal utilities [Modave and Grabisch, 1998]. This means that, unlike the attributes, which are defined on diverse scales and with different different units, it is possible to compare the satisfactions brought by the diverse criteria.

Given criteria i and j, and an alternative x, $u_i(x_i) \geq u_j(x_j)$ means that x is at least as satisfying on criterion i than on criterion j.

This is a very strong assumption, as expressed in [Grabisch and Labreuche, 2004]. Indeed, it might be hard for a DM to know how to understand, or interpret, that a surface area of 200 squared meters is as satisfying as having as a

price of €400,000. Nonetheless, it brings several benefits; in particular, it allows for comparison between criteria, which makes available some comparison-based operators, such as max and min, along with averaging operators.

In this regard, it is also usual to normalize the marginal utilities [Grabisch and Perny, 2003], such that $u_i(x_i) = 0$ means that x_i is totally disatisfying, and $u_i(x_i) = 1$ means that x_i is totally satisfying. Criterion i thus takes values in the unit interval.

We now present some models and methods from MAUT in order to build and represent preference relations. While our aim is not to be exhaustive, we give insight as to how such models are built in the state of the art.

2.3.2 Basic structure

The very essence of an MCDA model is to be able to represent preference relation $\succeq$ that fits that of the DM. In practice, let $M : X^2 \to \mathbb{R}$ a preference model. We also introduce a decision rule D, which determines whether the first argument of M is preferred or not to the second one from the output of M.

In crisp logic, D has output in $\{0, 1\}$ (or their boolean equivalent {True, False}). A common choice is a Heaviside function, where alternative a is said to be preferred to b if and only if $M(a, b) > 0$, and a is equivalent to b iff $M(a) = M(b)$. This is what is needed to represent a weak order as formalized in Definition 2.

Nonetheless, that requires perfect precision to model, as two models with the slightest difference in score (even imperceptible by a human decider) would mean a strict preference. As a consequence, it is not absurd to add a tolerance threshold ϵ to the decision model. Assuming we have a global utility function $\mathcal{F}$ defined on all alternatives in X, then we could have the following M and D:

$$M : (a, b) \mapsto \mathcal{F}(a) - \mathcal{F}(b)$$
$$D : t \mapsto \mathbb{1}_{\mathbb{R}_+^*}(t - \epsilon)$$

There are then three distinct cases:

- $\mathcal{F}(a) > \mathcal{F}(b) + \epsilon$; then $D(M(a, b)) = 1$ and $D(M(b, a)) = 0$: the model decides that a is preferred to b
- $\mathcal{F}(a) + \epsilon < \mathcal{F}(b)$; then $D(M(a, b)) = 0$ and $D(M(b, a)) = 1$: the model decides that b is preferred to a
- $|\mathcal{F}(b) - \mathcal{F}(a)| \leq \epsilon$; then $D(M(a, b)) = D(M(b, a)) = 0$; the model cannot decides that a and b are equivalent.

Note that, with this decision rule, we need to relax the transitivity of equivalence, as illustrated in Example 10.

Example 10.

- $\mathcal{F}(a) = \mathcal{F}(b) + \frac{2\epsilon}{3}$
- $\mathcal{F}(b) = \mathcal{F}(c) + \frac{2\epsilon}{3}$

then we have $a \sim b$ *and* $b \sim c$ *but* $a \succ c$ *strictly. This would violate the transitivity property. In this case,* $\succeq$ *is called a semi-order*[1].

In fuzzy logic, D can be a real function with its image being a real number between 0 and 1. Then it can be interpreted as the certainty that the first alternative is preferred to the second, or as the intensity of the preference, for instance. The choices of these functions depends on the problems we wish to solve and on the model itself.

Example 11. *Assume the following model:*

$$M : (a, b) \mapsto \mathcal{F}(a) - \mathcal{F}(b)$$
$$D : t \mapsto \max(0, \min(1, t))$$

This D *is illustrated in Figure 2.1. For instance, if* $\mathcal{F}(a) = 0.8$ *and* $\mathcal{F}(b) = 0.3$*, we have* a *preferred to* b *with a degree of* 0.5.

2.3.3 Aggregation and Comparison

An MCDA model such as described above thus takes values described by multiple criteria as input and yields a mono-dimensional result (either a boolean or numeric

[1] One could also imagine a model that is intermediate between the fuzzy model of example 11 and the crisp one described above; in this case, there could be several crisp thresholds $\epsilon_1 < ... < \epsilon_c$ such that:

- $\mathcal{F}(b) + \epsilon_2 > \mathcal{F}(a) > \mathcal{F}(b) + \epsilon_1$ means that a is *very weakly* preferred to b
- $\mathcal{F}(b) + \epsilon_3 > \mathcal{F}(a) > \mathcal{F}(b) + \epsilon_2$ means that a is *weakly* preferred to b
- ...
- $\mathcal{F}(b) + \epsilon_c > \mathcal{F}(a) > \mathcal{F}(b) + \epsilon_{c-1}$ means that a is *very strongly* preferred to b
- $\mathcal{F}(a) > \mathcal{F}(b) + \epsilon_c$ means that a is *extremely* preferred to b

This is the idea behind the `MACBETH` methodology [Bana e Costa and Vansnick, 1999], which utilizes 6 thresholds to build its preference model.

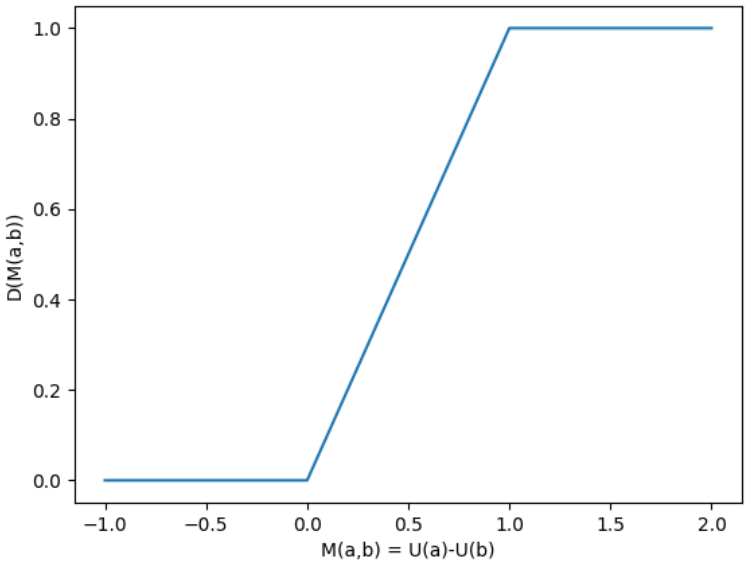


Figure 2.1: A fuzzy decision rule

value). This implies that, at some point, the information contained in all of the criteria is *aggregated* into this single, explicit one-dimensional value. Moreover, as it expresses preferences between two objects, there must also be a *comparison* step.

These are essentially the two core steps of an MCDA model. The order in which they are applied defines whether they are "compare-then-aggregate" or "aggregate-then-compare" models [Grabisch and Perny, 2003]. These families will be described in the next two sections.

2.3.3.1 Compare then Aggregate

Compare then Aggregate models have the following form :

$$M(a,b) = \mathcal{A}(\mathcal{C}_1(a_1,b_1),...,\mathcal{C}_n(a_n,b_n))$$

where each criterion i has a partial comparison operator on its performances $\mathcal{C}_i : X_i^2 \to \mathbb{R}$. $\mathcal{C}_i(a_i,b_i)$ is, of course, non-decreasing w.r.t. a_i and non-increasing w.r.t. b_i.

Then, $\mathcal{A} : \mathbb{R}^n \in \mathbb{R}$ is the aggregation operator. It indeed aggregates the n partial preferences into one global preference. $\mathcal{A}$ is non-decreasing w.r.t. its arguments.

Example 12. *The criterion-wise comparator of the performance on criterion i can thus have the shape:*

$$\forall a, b \in X, \mathcal{C}_i(a_i, b_i) = \mathbb{1}_{\{a_i > b_i\}} - \mathbb{1}_{\{a_i < b_i\}}$$

where $\mathbb{1}_S$ is the characteristic function of property S.
Then the aggregator $\mathcal{A}$ might be a simple sum:

$$\mathcal{A}(\mathcal{C}_1(a_1, b_1), ..., \mathcal{C}_n(a_n, b_n)) = \sum_{i=1}^{n} \mathcal{C}_i(a_i, b_i)$$

$M(a, b)$ thus yields the number of criteria where a dominates b minus the number of subjects where b dominates a. The decision rule is then

$$a \succ b \iff M(a, b) > 0$$

.

For instance, given houses from Example 1, the DM could compare houses h_1 and h_2 in a compare-then-aggregate fashion, by noticing that:

- *h_1 has a smaller surface area and a smaller garden than h_2*
- *h_1 has no garage, while h_2 has one*
- *h_1 is closer to the road, to downtown and to transportation systems than h_2*
- *h_1 is cheaper than h_2*

As a consequence, h_1 beats h_2 on 4 attributes $(4, 5, 6, 7)$ and is beaten by h_2 on the 3 other attributes. The aggregator $\mathcal{C}$ given above would yield that h_1 is preferred to h_2.
Nonetheless, a more complex aggregator could yield that the difference in price and accessibility is more than compensated by the presence of a garage and the bigger surface, and make h_2 the preferred alternative.

In this case, it is relevant to have skew-symmetric $\mathcal{C}_i$, along with a linear aggregator $\mathcal{A}$. We thus have $M(a, b) = -M(b, a)$ for each pair (a, b) of alternatives.

In this setting, ELECTRE method was first conceived by Roy in 1966 [Benayoun et al., 1966], and went on to be modernized through different versions (II, III, IV, TRI for instance). A comprehensive overview can be found in [Figueira et al., 2016]. These methods aim at solving problems of ranking, sorting and selecting the best between several alternatives. It functions by building preference relations and applying an elimination process on "unacceptable" alternatives, then dealing with the rest depending on the chosen problem.

2.3.3.2 Aggregate then Compare

Aggregate then Compare models have the following form :

$$M(a,b) = \mathcal{C}(\mathcal{A}(a_1,...,a_n), \mathcal{A}(b_1,...,b_n))$$

This time, both alternatives are aggregated independantly of each other, and the results of both aggregations are compared in order to determine the preferred alternative. Here as well, $\mathcal{A} : X \to \mathbb{R}$ is non-decreasing w.r.t. its arguments, and $\mathcal{C} : \mathbb{R}^2 \to \mathbb{R}$ is non-decreasing w.r.t. the first one and non-increasing w.r.t. the second one.

Example 13. *Assume we have a global utility $\mathcal{F}$ for scoring the houses. Then we can have the aggregation function $\mathcal{A}$ to be $\mathcal{F}$, and the comparison function $\mathcal{C}$ to simply be a difference. That is:*

$$M(a,b) = \mathcal{C}(\mathcal{A}(a), \mathcal{A}(b)) = \mathcal{F}(a) - \mathcal{F}(b)$$

with the same decision rule as in Example 12.

We see from Example 13 that, if we have a global utility $\mathcal{F}$ for the alternatives, $\mathcal{F}$ takes care of the aggregation step. It is thus very easy to work in the aggregate-then-compare framework.

This thesis focuses on this second setting. The problem is now the establishment of a global utility function which will be able to represent the preferences of the DM.

2.4 MAUT Models

2.4.1 Suitable Properties

We give here a list of important properties, given by [Grabisch and Perny, 2003] that we might expect our global utility functions to satisfy.

We call U the satisfaction space in which the values of the n criteria exist. If the criteria are commensurate and normalized, we can have $U = [0,1]^n$.

We recall that, for a given criterion i, $x_i = 0$ means that the alternative x is not satisfying at all w.r.t. criterion i, $x_i = 1$ means that it is, on the contrary, fully satisfying. In the same way, we say that $\mathcal{A}$ has its image in $[0,1]$, with 0 meaning that the alternative is not satisfying at all, while one means that it is fully satisfying.

We present below a list of properties which are often desirable for aggregation functions. Let **a** and **b** two alternatives in U:

- **Continuity**: w.r.t. all of the a_i
- **Monotonicity**: if, $\forall i \in N, a_i \geq b_i$, then $\mathcal{A}(\mathbf{a}) \geq \mathcal{A}(\mathbf{b})$
- **Piecewise-Linearity**
- **Idempotency**: $\forall t \in [0,1], \mathcal{A}(t,...,t) = t$

The first two are quite easily understandable. A continuous model is desirable for its stability and interpretability. This might even be reinforced by requiring that it be Lipschitz-continuous, in order to moderate the impact of the change of a criterion. Monotonicity is natural, given that the criteria already illustrate attribute-wise satisfactions. What it means is that, if an alternative is at least as satisfying as another one on all criteria, then it must be preferred.

Piecewise-linearity also allows some level of interpretation, by defining clear regions in which the model is linear, and thus simple to understand.

Idempotency states that, if an alternative **a** is equally satisfying on all criteria by a certain level, then it is also globally satisfying to the same level. Moreover, Idempotency and Monotonicity yield two more interesting properties of the model:

Compensatoriness: $\min(a_i) \leq \mathcal{A}(a_1,...,a_n) \leq \max(a_i)$

Compensatoriness is relevant as it makes the aggregator behave like an averaging operator, stating that an alternative cannot be more (resp. less) satisfying than its most (resp. least) satisfying criterion. This in turns allows to derive the normalization property:

Normalization: $\max_{\mathbf{a} \in U} \mathcal{A}(\mathbf{a}) = 1, \min_{\mathbf{a} \in U} \mathcal{A}(\mathbf{a}) = 0$

Normalization gives bounds to the possible satisfaction, yielding a maximal satisfaction level (with value 1) and a minimal one (with value 0).

2.4.2 Decomposable models

Under weak separability, we can assume a model $\mathcal{F}$, called a *decomposable models* [Krantz et al., 1971], that complies with the marginal preference relations.

Definition 3. *A MAUT model $\mathcal{F} : X \in [0,1]$ is called decomposable iff it can be written as:*

$$\mathcal{F}(\mathbf{x}) = \mathcal{A}(u_1(x_1),...,u_n(x_n)) \tag{2.3}$$

with $\mathcal{A}$ is non-decreasing w.r.t. its arguments and for each i in N, u_i a function which maps X_i into $\mathbb{R}$. The latter are the marginal utilities evoked in Section 2.3.1.1.

We present below a number of widely used aggregators, $\mathcal{F} : X \to \mathbb{R}$ step. Each time, we present how they are written, along with the requirements and properties. Note that they do not all fit the properties evoked in the previous sections.

2.4.3 Weighted sum

The weighted sum aggregator is one of the simplest. As such, it is used in many fields, from MCDA to everyday tasks. It can be written:

$$\begin{aligned} WS : X &\to \mathbb{R} \\ \mathbf{x} &\mapsto \sum_{i=1}^{n} w_i x_i \end{aligned}$$

Where the w_i are real weights. In order to ensure normalization, it is enough to assume that $\sum_{i=1}^{n} w_i = 1$. For monotonicity, we ensure $\forall i \in N, w_i \geq 0$.

This model is linear in all of its arguments. Each criterion has its own weight which represents to which extent increasing the satisfaction on the criterion affects the global score. Nonetheless, in order to obtain such an interpretation, it is required that the value on the attributes be commensurate and scaled. To achieve this, one might also assume that the weighted sum aggregates output of fixed marginal utilities $u_1, ..., u_n$, which serve as a rescaling and normalizing step:

$$\begin{aligned} WS : X &\to \mathbb{R} \\ \mathbf{x} &\mapsto \sum_{i=1}^{n} w_i u_i(x_i) \end{aligned}$$

As the weighted sum is a very simple model, it is easy to understand and interpret. Nonetheless, this comes with a very limited expressivity, as illustrated in Example 14.

Example 14. *We assume a sub-domain of Ex. 1, where we can evaluate houses only on criteria* 4, 5 *and* 6. *As we focus on the aggregation, we assume the marginal utilities to have already been computed. Thus, we represent a house as a vector of satisfactions on criteria* $4, 5, 6$*; that is, a house that is totally satisfying on criterion* 4 *and dissatisfying on criteria* $5, 6$ *is represented as* $(1, 0, 0)$. *Let* $\succ$ *and* $\sim$ *be a preference relation and an equivalence relation on houses restricted to criteria* $4, 5, 6$.

The DM considers that public transportation will mostly be used for accessing the downtown hub; it is thus equally satisfying to be close to downtown, to the

public transportation, or to both. This means that, given the satisfaction α on criterion 4, we have

$$(\alpha, 0, 1) \sim (\alpha, 1, 0) \sim (\alpha, 1, 1). \tag{2.4}$$

The DM also considers that having only access to the city (5. or 6.) is as satisfying as having only access to the outside world (4.), but is still better than having access to neither. Thus

$$(0, 1, 0) \sim (1, 0, 0) \succ (0, 0, 0). \tag{2.5}$$

Given a weighted sum, first equation gives us $w_5 = w_6 = 0$, and thus $w_4 = 1$. On the other hand, the second one gives $w_4 = w_5$; we can thus not represent the DM's rules with a weighted sum.

2.4.4 Ordered weighted average

Introduced in 1988 by Yager [Yager, 1988], the ordered weighted average operator is written:

$$\begin{aligned} OWA : X &\to [0, 1] \\ \mathbf{x} &\mapsto \sum_{i=1}^{n} w_i x_{\tau(i)} \end{aligned}$$

where the w_i are weights, and $\tau(i) : N \to N$ is a permutation of the indices of the criteria so that $x_{\tau(1)} \geq x_{\tau(2)} \geq ... \geq x_{\tau(n)}$.

This model is very similar to the weighted sum, as the weights also sum to one and are positive in order to ensure monotonicity and normalization. The difference with the weighted sum is that each weight is associated not with a given criterion index, but with the rank of the value on a criterion among all the values on all of the criteria.

As a weight-based model, it is thus also necessary that the values be commensurate, so that the weights make sense; indeed, the same weight can be applied to any of the attributes, depending on their values.

This particular type of aggregator can be used in contexts where we are given a set of items to evaluate, whose order is not relevant. For instance, one could evaluate the relevance of a list of movies suggested by an online streaming service, with the attributes being the individual relevance of each movie w.r.t. the current customer.

2.4.5 Additive Utilities

The additive utilities model [Fishburn, 1967a] is a generalization of the weighted sum in the context of MAUT. It is written in the following way:

$$\begin{aligned} AU : X &\to [0,1] \\ x &\mapsto \sum_{i=1}^{n} v_i(x_i) \end{aligned}$$

where the $v_i : X_i \to \mathbb{R}_+$ are local utility functions, associated with each criterion, and yielding the marginal satisfaction brought by x on each criterion. Normalization is ensured through the fact that:

$$\sum_{i=1}^{n} v_i(T_i) = 1$$

where $T_i \in X_i$ is the element that maximizes v_i (i.e. the "best" element). As [Siskos, 1982] states, one can re-write each v_i as:

$$\forall i \in N,\ \forall x_i \in X_i,\ v_i(x_i) = w_i(x_i) \text{ with } w_i = v_i(T_i)$$

We thus have weights w_i, along with normalized marginal utilities u_i as evoked before.

2.4.6 Generalized Additive Independance

Generalized additive independance (GAI) was introduced by Fishburn in 1967 [Fishburn, 1967b]. It generalizes the Additive Utility model described above, as it allows the utility functions to be multidimensionnal.

Formally, we have:

$$\begin{aligned} GAI : X &\to [0,1] \\ x &\mapsto \sum_{S \subseteq 2^N} u_S(\mathbf{x}_S) \end{aligned}$$

where $\mathbf{x}_S$ is the orthogonal projection of x on S (only the criteria belonging to S are kept), and $u_S : \prod_{i \in S} X_i \to [0,1]$ are utility functions for some subsets S of N.

This model is more general than the additive utility (if we only have utilities u_S for $|S| = 1$, then the model is the AU).

This generality, as described above, can prove problematic if it is excessive w.r.t. the complexity of the real decision process (that of the DM). In particular, this model without restriction violates preferential independence weak separability [Grabisch and Labreuche, 2015]. Thus, this model is clearly not decomposable. Moreover, in the general case, it might require up to an exponential (w.r.t. n)

	Additive	Non-Additive
Weights	Weighted Sum, OWA	Choquet Integral
No Weights	Additive Utility	GAI

Table 2.3: Relation Between the Aggregators

number of marginal utilities to be specified, which quickly becomes computation-heavy.

In this section, we present an aggregation function, called the Choquet integral [Choquet, 1954], which is at the heart of this thesis's contribution. This model is interesting, as it generalizes the weighted sum, is compatible with marginal utilities for building a decomposable model, and has many interesting properties required in the previous sections. It is in particular more constrained and interpretable than a GAI [Grabisch and Labreuche, 2018], and more expressive than most additive models, thus yielding a nice compromise between both families.

Table 2.3 shows the relation between the previously cited aggregators.

2.5 Fuzzy Measures

We describe below a way to take into account interactions among criteria, while keeping, as much as is possible, the simplicity (and thus interpretability) of linear models such as the weighted sum. We present here *fuzzy measures* [Choquet, 1954, Sugeno, 1974, Grabisch and Labreuche, 2004] to generalize the notion of weights, in that purpose.

2.5.1 Definition

Let $P = 2^N$ the set of all subnets of N. A *fuzzy measure*, or *capacity* on N is a set function $\mu : P \to [0, 1]$ which satisfies the following three properties:

- **Normal1:** $\mu(\emptyset) = 0$
- **Normal2:** $\mu(N) = 1$
- **Monot:** $\forall A \subseteq N, \forall B \subseteq A, \mu(B) \leq \mu(A)$

These properties are easy to understand. **Normal1** and **Normal2** ensure normalization of the measure; that is, an empty set has a zero measure, the complete set has a 1 measure. On the other hand, **Monot**ensures that all sets have a larger measure than any of their subsets.

In a weighted sum, there are n weights, which represent a discrete, additive measure on N. That is, all attributes have a given weight, and the weight of a set of attributes (assimilable to its impact on the output of the model) is the sum of the weights of its elements. This is represented by a fuzzy measure μ such that, $\forall S \subseteq N,\ \mu(S) = \sum_{i \in S} \mu(\{i\})$. In the more general case of a fuzzy, non-additive measure, this is not true anymore, as there is now a "weight" (i.e. its measure) for any set, or coalition, of criteria, with the constraint being that no criterion contributes negatively to a coalition (i.e. adding an element to a coalition cannot decrease its weight, by **Monot**).

This principle is widely used in MCDA [Grabisch, 1996, Grabisch et al., 2000] as it allows the modelling certain types of interactions among criteria [Grabisch and Labreuche, 2004] by varying the weight assigned to this very subset of criteria. To some extent, for $S \subseteq N$, $\mu(S)$ can be seen as a measure of the importance of coalition S within the whole set N.

2.5.2 Möbius transform of a fuzzy measure

The Möbius transform [Rota, 1964, Grabisch, 1999] m of fuzzy measure μ is another, equivalent, way of writing μ. Formally, m is the only function that satisfies:

$$\forall S \subseteq N, \mu(S) = \sum_{B \subseteq S} m(B) \tag{2.6}$$

m can be explicitly written from μ as:

$$\forall S \subseteq N, m(S) = \sum_{B \subseteq S} (-1)^{|S \setminus B|} \mu(B)$$

The Möbius of a coalition $m(S)$ can be seen as the importance, or weight, of this coalition "on its own", regardless of the importance of its sub-coalitions. In particular, while for $S \subseteq N$, $\mu(S)$ is in the unit interval, $m(S)$ is in $[-1, 1]$. The normalization and monotonicity contstraints on an FM μ can be translated in terms of the Möbius transform:

- **Normal1:** $m(\emptyset) = 0$
- **Normal2:** $\sum_{S \subseteq N} m(S) = 1$
- **Monot:** $\forall A \subseteq N, \forall B \subseteq A, \sum_{S \subseteq A, S \not\subseteq B} m(S) \geq 0$

2.5.3 Representation of Preferences by a Fuzzy Measure

Assume a context where a criterion is either not satisfied (value $\mathbb{0}$) or totally satisfied (value $\mathbb{1}$). An alternative is thus an element of $\{\mathbb{0}, \mathbb{1}\}^n$. For $S \subseteq N$, we write $\mathbf{a}_S$ the alternative such that $\mathbf{a}_S = (\mathbb{1}_S, \mathbb{0}_{-S})$ is satisfying on all criteria in S, and only on those.

Let μ a fuzzy measure on N, we can measure the satisfaction of an alternative $\mathbf{a}_S$ as $U((\mathbb{1}_S, \mathbb{0}_{-S})) = \mathcal{A}((1_S, 0_{-S})) = \mu(S)$; that is, the satisfaction on a is the measure of the set of criteria on which a is satisfying.

Then, the properties become intuitive. They can be translated as:

- **Normal1:** the alternative which is not satisfying on any criteria is not satisfying at all
- **Normal2:** the alternative which is totally satisfying on all criteria is totally satisfying
- **Monot:** if an alternative $\mathbf{a}_A$ is at least as satisfying on all criteria than an alternative $\mathbf{a}_B$, then $\mathbf{a}_A$ is globally at least as satisfying as $\mathbf{a}_B$

This notion will be generalized to intermediate values for the criteria.

We stated above that fuzzy measures are used to represent interactions among criteria. We now formalize this concept of interaction following [Grabisch, 1997a, Grabisch and Labreuche, 2004].

Considering two coalitions of criteria $A, B \subseteq N^2$, with $A \cap B = \emptyset$ there are three possible cases:

- **Redundancy:** $\mu(A \cup B) < \mu(A) + \mu(B)$
- **Independance:** $\mu(A \cup B) = \mu(A) + \mu(B)$
- **Synergy:** $\mu(A \cup B) > \mu(A) + \mu(B)$

In the first case, the union of the coalition brings less value than the sum of the values of the coalitions taken separately. The coalitions are thus considered as *redundant*, as some of the value they bring is common to them both.

In the second case, both coalition together bring the same value as when they are taken separately. They are thus considered independent, as there is no interaction between the values brought by both coalitions.

Finally, in the last case, getting both coalitions together brings some extra value compared to what they bring independently. This phenomenon is called *synergy*, or *complementarity*.

2.5.4 k-additive fuzzy measure

A fuzzy measure μ, or equivalently its Möbius representation m, has one value for each of the subsets of N. This means that it has 2^n values, which might lead to several problems when using one to model a DM behaviour:

- exponential computational cost
- too many degrees of freedom can make a model too "loose", leading to "ghosts in the data", a.k.a. overfitting (we shall return to this issue in Section 3.6.1)
- a very hard time interpreting the model

In order to limit complexity of the models based on fuzzy measures, Grabisch introduced the notion of *k-additivity* [Grabisch, 1997b]. Let $k \in [1, n]$. A fuzzy measure μ is said to be *k-additive* if and only if:

$$\forall S \text{ s.t. } |S| > k,\ m(S) = 0$$

This property ensures that the model does not take into account the interactions between more than k criteria. It thus simplifies the model, by making it more rigid, as it lowers the number of values needed from 2^n to $O(n^k)$.

In practice, in MCDA, 2-additive measures are usually enough used to parameterize a model. These allow to represent interactions between pairs of criteria, while forbidding interactions between sets of three or more criteria. In practice, this flexibility is sufficient to model adequately the DM's preference function. Indeed, while humans can easily acknowledge interactions between two criteria, they usually have a hard time in trying to take into account interactions between three criteria or more. Allowing more degrees of freedom to the model would just add extra, non-interpretable parameters which, while making the model richer, require more information to characterize.

2.5.5 Shapley values and interaction indices

In general, as we have seen, and without any assumption on its additivity, it might be difficult to quickly analyze the behaviour represented by a fuzzy measure μ.

It is thus relevant to have indicators, computed from a fuzzy measure, which allows to interpret diverse behavioral elements of the fuzzy measure at a glance.

2.5.5.1 Shapley Values

Shapley values [Shapley, 1953] were introduced in collaborative game theory. In these types of games, a coalition of players work together in order to obtain a global reward. Then comes the question of how to split the reward among the players in a way that reflects the respective contribution of each player.

While distributing the reward equally among the players is the easiest way to share it, it might not be representative of how much any of the players brought to the coalition.

In the case of a fuzzy measure, the analogy is trivial: the n criteria are players, which collaborate in order to maximize their reward, i.e. the global utility of the alternative. In this case, the contribution of a criterion is analogous to its importance in the model, i.e. how much it influences the final output.

In this regard, we want to compute the average impact that a criterion i has on the output of the model. Analogously, we assume that a criterion can either be satisfied (value 1) or not satisfied (value 0). Then, for an alternative $\mathbf{x} \in X$, we call $S \subseteq N$ the set of criteria on which $\mathbf{x}$ is satisfying. A logical way to evaluate the impact of a criterion $i \notin S$ would be to compare the utility of two alternatives whose criteria are $\mathbf{a}_S = (1_S, 0_{-S})$ and $\mathbf{b}_S = (1_{S\cup\{i\}}, 0_{-S\cup\{i\}})$. That is, we check the difference between the utility of an alternative where criterion i is not met, and that where it is satisfied.

Let $\mathcal{A} : [0,1]^n \to [0,1]$ a normalized aggregation model. If $\mathcal{A}$ is an additive model, say, for instance, a weighted sum, then $\mathcal{A}$ can be written as:

$$\mathcal{A}(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i$$

As this model is additive, the difference $\mathcal{A}(b_s) - \mathcal{A}(a_s)$ is always w_i, independently of S. Criterion i thus contributes to a degree that is proportional to its weight w_i.

This makes the weights suitable measures for the contribution of their respective criteria. In particular, as the weights sum up to one by definition of the model, such weights form a relevant distribution of the contribution to attribute to each criterion.

This becomes more complicated when the model is not additive; that is, we do not necessarily have:

$$\mathcal{A}(\mathbf{b}_{S_1}) - \mathcal{A}(\mathbf{a}_{S_1}) = \mathcal{A}(\mathbf{b}_{S_2}) - \mathcal{A}(\mathbf{a}_{S_2})$$

In particular, if we assume the fuzzy-measure based μ introduced in Section 2.5.3, we can consider that the average contribution of being satisfied on i is the average of the terms of shape $\mu(a_{S\cup\{i\}}) - \mu(a_S)$ on any $S \subseteq N \setminus \{i\}$. This is

the idea behind the computation of the Shapley value. For a criterion i, and a fuzzy-measure μ the Shapley value Φ_i^μ is given by [Grabisch and Labreuche, 2004]:

$$\Phi_i^\mu := \sum_{S \subseteq N \setminus \{i\}} \frac{(n - |S| - 1)!|S|!}{n!} [\mu(S \cup \{i\}) - \mu(S)] \tag{2.7}$$

Once computed, the Shapley values form an n-dimensionnal vector $(\Phi_1^\mu, ..., \Phi_n^\mu)$, which has the interesting property that:

$$\sum_{i \in N} \Phi_i^\mu = 1$$

It easy to see that all of the Φ_i^μ are non-negative as μ is monotonic. Thus, the Φ_i^μ can be seen as a distribution of importance on all criteria. Note that, if μ is 1-additive, it is a set of weights $(w_1, ..., w_n) = (\mu(\{1\}), ..., \mu(\{n\}))$, and its Shapley values $\Phi_i^\mu = w_i$.

This Shapley value is very important in MCDA, as it allows to quickly check which criteria have a greater impact on a model [Murofushi and Soneda, 1993, Labreuche and Fossier, 2018]. Nonetheless, as one sees, they require, in the general case, an exponential number of computations. This can be simplified for k-additive fuzzy measures.

2.5.5.2 Interaction Indices

While the Shapley values are great at indicating the comparative importance of the criteria, they miss a type of information that is critical when analyzing a fuzzy-measure-based model; that is, it leaves unexplained the interaction part.

Thus, in the same idea than the Shapley values, [Murofushi and Soneda, 1993] proposes the following interaction index for studying the type and relative importance of the interaction among a pair of criteria i and j:

$$I_{i,j}^\mu := \sum_{S \subseteq N \setminus \{i,j\}} \frac{(n - |S| - 2)!|S|!}{(n-1)!} [\mu(S \cup \{i,j\}) - \mu(S \cup \{i\}) - \mu(S \cup \{j\}) + \mu(S)] \tag{2.8}$$

This index was later generalized by Grabisch in [Grabisch, 1997b] in order to reflect the interaction for any coalition $A \subseteq N$ [Grabisch and Labreuche, 2004]:

$$I^\mu(A) := \sum_{S \subseteq N \setminus A} \frac{(n - |S| - |A|)!|S|!}{(n - |A| + 1)!} \sum_{L \subseteq A} (-1)^{|A \setminus L|} \mu(S \cup L) \tag{2.9}$$

Note that, in the case of a 1-additive fuzzy measure, the interaction indices are all 0. In the case of a 2-additive one, they are the Möbius of the corresponding pair.

Set	μ	m
$\emptyset$	0	0
$\{4\}$	0.5	0.5
$\{5\}$	0.5	0.5
$\{6\}$	0.5	0.5
$\{4,5\}$	1	0.
$\{4,6\}$	1	0.
$\{5,6\}$	0.5	-0.5
$\{4,5,6\}$	1	0.

Table 2.4: μ, defined on all subsets, along with its Möbius values

2.6 Choquet Integral

2.6.1 Definition

The Choquet integral [Choquet, 1954] is a particular case of the GAI model described in 2.4.6, and a generalization of the weighted sum. It is parameterized by a fuzzy measure μ, and is written as:

$$\begin{aligned} C_\mu : X &\to \mathbb{R} \\ \mathbf{x} &\mapsto \sum_{i=1}^{n} \left(x_{\tau(i)} - x_{\tau(i-1)}\right) \mu(A_i) \end{aligned} \tag{2.10}$$

where $\tau(i) : N \to N$ is a permutation of the indices of the criteria so that $x_{\tau(1)} \geq x_{\tau(2)} \geq ... \geq x_{\tau(n)}$, $x_{\tau(0)} = 0$, and $A_i = \{\tau(i), \tau(i+1), ..., \tau(n)\}$.

If we use the Möbius representation m of μ, we can alternatively write it in the equivalent form:

$$\begin{aligned} C_\mu : X &\to \mathbb{R} \\ \mathbf{x} &\mapsto \sum_{S \in N} m(S) \min(\{x_i | i \in S\}) \end{aligned} \tag{2.11}$$

As we manipulate the attributes through min operators, along with computing their differences, it is essential for interpreting this model that the inputs (the x_i) be commensurate.

Example 15 (Ex. 14 cont.)**.** *We define the fuzzy measure μ on set $\{4,5,6\}$, with its values given in Table 2.4.*

As these values are not necessarily commensurate, we assume that we work on a normalized version of these attributes, i.e. the satisfaction they each bring. In this case, the FM is 2*-additive, meaning that its interaction index of a pair of indices is the Möbius value of said pair. It is thus easily interpretable from the third column.*

In particular, we see that the term $m(\{4,5\}) = -0.5$ *indicates some redundancy among both attribute* 5 *(satisfaction on the distance to the public transportation system) and attribute* 6 *(satisfaction on the distance to downtown). The Shapley values are also easily computable:*

$$\Phi_4 = 0.5; \Phi_5 = \Phi_6 = 0.25$$

From Equation (2.11)*, the equation can be simplified as:*

$$\begin{aligned} C_\mu(\mathbf{x}) &= \frac{1}{2}(x_4 + x_5 + x_6) - \frac{1}{2}\min(x_5, x_6) \\ &= \frac{1}{2}(x_4 + \max(x_5, x_6)) \end{aligned}$$

As one can see, being satisfied on the proximity to a road (attribute 4*.) brings only half of the maximal satisfaction. The other half is brought by how accessible the city is, either through public transportation (criterion* 5*) or by being close to downtown (criterion* 6*). Nonetheless, only one of the latter is enough (as represented by the* max *term).*

A Choquet integral parameterized by μ *fulfills the preferences given in Ex. 14 – namely relations (2.4) and (2.5).*

Let us compute

$$C_\mu(\alpha, 0, 1) = \alpha\,(\mu(\{4,6\}) - \mu(\{6\})) + 1\,\mu(\{6\}) = \frac{\alpha}{2} + \frac{1}{2}.$$

Likewise, $C_\mu(\alpha, 1, 0) = \frac{\alpha}{2} + \frac{1}{2} = C_\mu(\alpha, 1, 1)$*, so that condition (2.4) is satisfied. We also check that:*

$C_\mu(0, 1, 0) = \mu(\{5\}) = \frac{1}{2} = C_\mu(1, 0, 0) > C_\mu(0, 0, 0) = 0.$

Hence (2.5) is also satisfied. This Choquet integral allows to represent the preferences of the DM, unlike any possible weighted sum.

Its ability to rationally model a vast array of preference processes [Murofushi and Sugeno, 1989] has made it one of the most widely used models in MCDA [Grabisch and Labreuche, 2008, Grabisch, 2006]. In particular, there has been some notable work on describing and adapting the model to different axiomatics and conditions sets [Mayag et al., 2011, Labreuche, 2018]. A survey of more recent

methods and uses of the Choquet Integral can be found in [Beliakov and Divakov, 2021].

The Choquet integral offers many interesting properties, due to its structure. Regardless of its parameterization (the fuzzy measure), a Choquet integral satisfies the properties cited in 2.4.1.

2.6.2 2-Additive Choquet Integral

As it is parameterized by a fuzzy measure, the Choquet Integral (CI) might also suffer from excessive flexibility. This can be easily tackled by using a k-additive fuzzy measure for our model, in order to reduce the model complexity to a polynomial one.

In particular, as evoked above, 2-additive fuzzy measures are frequently chosen to represent the decision process of a human DM. The thus-called 2-additive Choquet Integral can then be written in the following way (parameterized here by the Möbius representation of the fuzzy measure μ):

$$\begin{aligned} C_m : X &\to [0,1] \\ x &\mapsto \sum_{i=1}^{n} m(\{i\})x_i + \sum_{i=1}^{n}\sum_{j=i+1}^{n} m(\{i,j\})\min(x_i, x_j) \end{aligned}$$

One can easily see this as a weighted sum, to which is added a second term containing the interactions between pairs of criteria. It is to be noted that a Choquet integral parameterized by a 1-additive (also simply called *additive*) fuzzy measure is precisely a weighted sum.

2.7 Hierarchical Models

2.7.1 Motivation

When the number of criteria increases, so does the number of parameters. Models can then become difficult to interpret by the decision maker. A common practice in multi-criteria decision therefore is to structure the criteria along a hierarchy. Formally, subsets of criteria are grouped into so-called *intermediate* or *artificial* criteria. These intermediate criteria are iteratively aggregated (possibly with initial criteria) until yielding a single score.

Definition 4. *A hierarchy on N consists of a directed rooted tree $\mathcal{T} = \langle r, M, \mathrm{Ch}\rangle$ where M is the set of vertices (leaves included). Let V and N respectively denote the set of non-leaf nodes and the set of leaves (as each leaf corresponds to a native*

criterion), with thus $M = V \cup N$. $r \in V$ *is the root node (the top aggregation node) and* $\mathrm{Ch} : M \to 2^V$ *is the set of children of every node.*

For $k \in V$, let $\mathrm{Lf}(k)$ denote the set of leaves in the subtree of $\mathcal{T}$ rooted at k. We also write $\mathbf{x}_k$ (resp. X_k) the restriction of $\mathbf{x}$ (resp X) to the attributes in $\mathrm{Lf}(k)$.

For $k \in V$, let $d(k)$ denote the number of children nodes of k: $d(k) = |\mathrm{Ch}(k)|$. We suppose the children to be ordered by their index in M, and we write $\mathrm{Ch}(k) = \{k_1, ..., k_{d(k)}\}$.

Example 16. *Assume the problem and criteria given in Example 1. These criteria are gradually aggregated as illustrated in Fig 2.2, to form new compound criteria:*

8. *Commodities, aggregates Garden and Garage*
9. *Building comfort, aggregates Surface and Commodities*
10. *Accessibility, aggregates distances to Road, Transportation and Downtown*
11. *Global score, aggregates Comfort, Accessibility and Price*

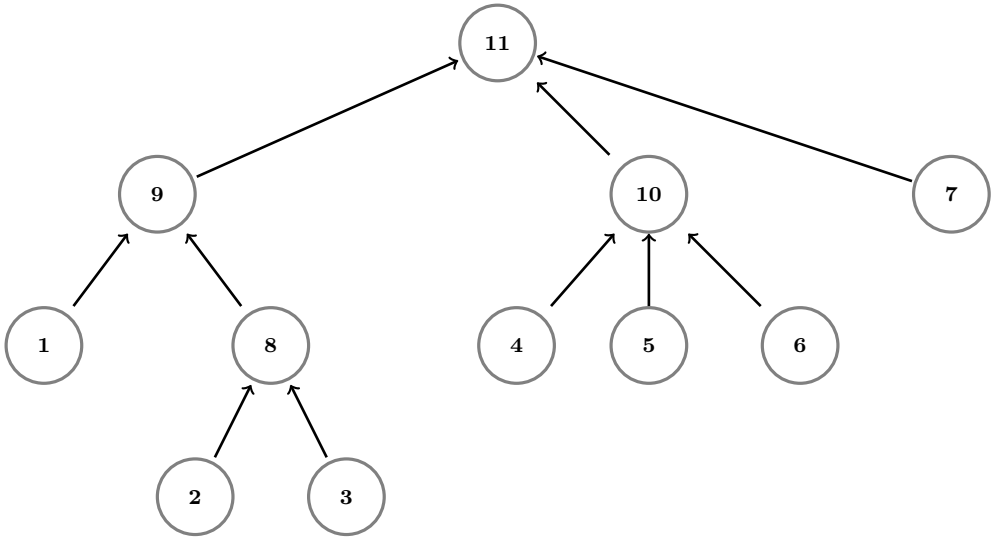


Figure 2.2: Hierarchy of criteria described in Ex 1.

Definition 5. *A hierarchical model* $\mathcal{F} = \langle \mathcal{T}, \nu \rangle$ *is composed of a hierarchy* $\mathcal{T} = \langle r, M = V \cup N, \mathrm{Ch} \rangle$ *and of a set of aggregators* $\nu = \{\mathcal{A}_k \mid k \in V\}$*, such that all non-leaf nodes of* $\mathcal{T}$ *have an associated aggregator.*

For all nodes $k \in V$, by construction,the number of values aggregated in $\mathcal{A}_k$, also referred to as the dimension of $\mathcal{A}_k$, is equal to $\mathrm{Ch}(k)$.

Let $\mathbf{x} \in X$. We assume that each node $i \in M$ has a certain utility $a_i(\mathbf{x})$. The a_i are computed recursively from the leaves as:

$$a_i(\mathbf{x}) = \begin{cases} \mathcal{A}_i((a_k, k \in \mathrm{Ch}(i))) & \text{if } k \in V \\ x_i & \text{otherwise} \end{cases}$$

That is: each node aggregates the output of its children. The output of the model is a_r, the output of the aggregation at the root node.

This is illustrated in Example 17.

Example 17. *In Figure 2.2, nodes 1 to 7 are native criteria, and nodes 8 to 11 are aggregation nodes. Their values are computed as the aggregation of their respective children in the tree, which can be either criteria or aggregation nodes. In particular, Accessibility (node 10) will aggregate the proximity to a large road, to downtown and to public transportation systems (respectively nodes 4, 5, and 6).*

Compared to models with a single aggregation, thereafter referred to as *flat models*, hierarchical models offer several advantages:

- Representation power: due to the non-linearity of the aggregation, hierarchical models are strictly more representative than their flat counterpart, as a flat model is simply a 1-level hierarchy;
- Sparsity: if the number of parameters of an aggregator is more than linear, then a hierarchical model involves fewer parameters than a flat one with the same number of criteria; likewise, the grouping of criteria into a smaller number of intermediate ones makes it easier for the decision maker to interpret the mode (see Example 18);
- Representation of domain knowledge: hierarchical models can be tailored to reflect pieces of expert knowledge and thus better fit the mental representation of the decision maker;
- Hierarchical structures can be inspected at different levels of detail: if the decision maker needs to better understand the value of a given intermediate criterion, they can look at the values on its children nodes and get a fine-grained explanation thereof [Labreuche and Fossier, 2018].

Example 18 (Ex. 1 cont.)**.** *Consider the tree in Fig. 2.2. With 2^k parameters for an aggregation node (a non-additive Choquet Integral, for instance) with k children, we have $4+4+8+8 = 24$ parameters in a model with that hierarchy. On the other hand, a flat model, with a single aggregation, would have $2^7 = 128$ parameters. The number of parameters is thus greatly reduced by such a hierarchy.*

2.7.2 Hierarchical Choquet Integral

2.7.2.1 HCI Model

The hierarchical Choquet Integral (HCI) model retains the generalization from a single CI to a network of CIs. Nonetheless, it restricts their structure to a tree. That is, it is exactly a hierarchical model as defined in Section 2.7, with CIs as the aggregators.

Each leaf of the tree corresponds to a single criterion, and each criterion appears as a single leaf. Accordingly, the non-leaf nodes in the tree have a clear meaning to the DM, as illustrated in Ex. 16. Note that, whatever the structure of the tree, the number of aggregations and the size of each aggregation is at most $n-1$. Therefore, the number of parameters is lower or equal to that of a CI.

Each non-leaf node $k \in V$ has an associated FM μ_k on the set $\mathrm{Ch}(k)$. A *hierarchical Choquet integral* (HCI) on $\mathcal{T}$ is the set of hierarchical functions $\mathcal{A} = \{\mathcal{A}_k\}_{k\in M}$ where $\mathcal{A}_k : Y^{|\mathrm{Lf}(k)|} \to Y$ is computed recursively as:

$$\mathcal{A}_k(\mathbf{a}) = C_{\mu_k}\Big(\mathcal{A}_{k_1}(\mathbf{a}_{k_1}), \ldots, \mathcal{A}_{k_{d(k)}}(\mathbf{a}_{k_{d(k)}})\Big)$$

if $k \in V$, and $\mathcal{A}_k \equiv \mathrm{Id}$ if $k \in N$.

HCIs generalize CIs, preserving all CI properties, and bringing along additional benefits. Essentially, the hierarchical decomposition involves many small, and thus highly interpretable, aggregations (as opposed to a single, big one). This decomposition is most appreciated for large n values to avoid cognitive fatigue, as humans can hardly keep more than 7 elements in mind at the same time, according to Miller's law [Miller, 1956]. The reduced number of parameters also helps to learn such models, by limiting overfitting.

This also allows for a local analysis of only the necessary parameters. In Example 1, if two houses have similar evaluations, the DM might wish to look at the direct nodes, and see that one offers more ease of access, while the other is much more comfortable. Nonetheless, if two artificial criteria still have similar values, the DM can "focus" on the associated node, and get a finer explanation, and so on, going as deep as they want into the hierarchy. Finally, the normalization and monotonicity constraints are still valid, as a composition of CIs. Due to the compensatoriness property, a CI on a single criterion is the identity function. We thus impose that every non-leaf node k has at least two children ($d(k) \geq 2$).

2.7.3 Global Winter Values

A hierarchical version of the Shapley value, called the global Winter value was also established for analyzing such models [Winter, 1989, Labreuche and Fossier, 2018]. We give here their definition.

Definition 6. *Let $k \in M$ a node of the hierarchy. Let $\pi_k = \{r = p_1, ..., p_m = k\} \subseteq M$ the path from the root to k. That is, $\forall i \in \{2, ..., k\}$, $p_i \in \mathrm{Ch}(p_{i-1})$. For a node $v \in V$, and $j \in \mathrm{Ch}(v)$, we write $\Phi_j^{\mu_v}$ the Shapley value of j w.r.t. the FM that parameterizes its parent v. In other words, $\Phi_j^{\mu_v}$ is the importance of j in the aggregation v, relative to that of the other children of v.*

*Then, the **global Winter value** Ψ_k of node k is defined as:*

$$\Psi_k = \prod_{i=2}^{k} \Phi_{p_i}^{\mu_{p_{i-1}}} \tag{2.12}$$

It is easy to see that, on a flat model, this corresponds exactly to the Shapley value. We only work with global Winter values (as opposed to their local version) later on; we thus use simply *Winter value* later on.

2.7.4 Utilitaristic HCI Model

Definition 7. *Let $\mathcal{T}$ be a hierarchy with root r on the set of criteria N. Let*

- *$\mathcal{U} : X \to U$ be a function, composed by n marginal utilities $\{u_1, \cdots, u_n\}$, one on each native attribute i.e. only on the leaves. With the above notation, $\mathcal{U}(\mathbf{x}) = (u_1(x_1), \cdots, u_n(x_n))$, thus, $\mathcal{U}$ maps an alternative $\mathbf{x}$ to the vector corresponding to $\mathbf{x}$'s criteria-wise satisfaction in the utility space.*
- *$\mathcal{A} : U \to Y$ be an HCI.*

Function $\mathcal{F} : X \to Y$ defined by $\mathcal{F} = \mathcal{A} \circ \mathcal{U}$ is called a Utilitaristic Hierarchical Choquet Integral (UHCI). *Applications of such models can be found in [Angilella et al., 2013].*

The utilities are computed recursively from the leaves to the root node.

Given $\mathbf{x}_N \in X$, we first compute the utilities on the criteria: $a_i = u_i(x_i)$, for $i \in N$ on the leaves. Then the utility at node $k \in V$ is given by $a_k = \mathcal{F}_k(\mathbf{x}_k) = \mathcal{A}_k(\mathbf{a}_{\mathrm{Ch}(k)})$, where it aggregates the utility values of its children. Finally the overall utility is the utility a_r at the root node: $a_r = \mathcal{F}_r(\mathbf{x}_N) = \mathcal{F}(\mathbf{x}_N)$.

Example 19. *[Ex. 15 cont.] The recursive computation of the utility values in the tree of Fig. 2.2 is done as follows:*

- $a_1 = u_1(x_1), \ldots, a_7 = u_7(x_7)$,
- $a_8 = C_{\mu_8}(a_2, a_3)$,
- $a_9 = C_{\mu_9}(a_1, a_8)$,
- $a_{10} = C_{\mu_{10}}(a_4, a_5, a_6)$,

- $a_{11} = C_{\mu_{11}}(a_9, a_{10}, a_7)$.

Note that Ex. 15 gives an illustration of the computation of a_{10} *given the value of its children* a_4, a_5, a_6. *Here* μ_{10} *is the FM given in Ex. 15. The overall score is then the score* a_{11} *of the root node* 11.

Assume a house defined on $(4), (5), (6)$ *by the values* $(1.5, 10, 3)$ *in kilometers. Our scoring model* $\mathcal{F}$ *is a UHCI composed by:*

- u_4, u_5, u_6, *a marginal utility for each criteria*
- $\mathcal{A}$, *an HCI composed by a single aggregation parameterized by the FM given in Ex. 15.*

Assuming we have $u_4(1.5) = 0.8$, $u_5(10) = 0.0$, $u_6(3) = 0.5$, *our final score is computed through* (2.10) *as:*

$$\begin{aligned}\mathcal{F}(x_4, x_5, x_6) &= \mathcal{A}(0.8, 0.0, 0.5)\\ &= 0.5\,(\mu(4,6) - \mu(4)) + 0.8\ \ \mu(4)\\ &= 0.65\end{aligned}$$

2.8 Deterministic Elicitation of MCDA models

After presenting the models the models involved in this manuscript, let us describe the existing approaches to build such models. These methods, usually rooted in Operations Research, are mostly based on combinatorial optimization. Moreover, they often require an interaction with the DM in order to ensure the absence of any contradiction in the preferential information (for instance, a triangular preference $a \succ b \succ c \succ a$).

2.8.0.1 General models

The ELECTRE methods [Benayoun et al., 1966] rely on the intensive interaction with the DM, manually adjusting some (sensitive) hyper-parameters of the algorithms. This limitation is alleviated respectively in the ORESTE and PROMETHEE approaches. The ORESTE methodology [Roubens, 1982, Pastijn and Leysen, 1989] inverses the order of the ELECTRE method, first building a complete preorder, then invalidating some parts of the preorder (marking them as incomparable). The PROMETHEE method [Brans and Vincke, 1985] uses graphs parameterized by intelligible parameters in order to build a total preorder on the set of possible actions (thus a preference relation on the said set, allowing ranking tasks).

Some work was done on elicitating the parameters of Majority Rule Sorting (MR-Sort), a simplified ELECTRE TRI model through mixed-integer programming [Leroy et al., 2011, Sobrie et al., 2016]. The same methods were also used to learn 2-additive non compensatory models by Sobrie [Sobrie et al., 2015].

Mixed integer programming was also applied to elicit the parameters of the Borda count [Benabbou et al., 2016a], a weighted vote method.

2.8.0.2 Utility-functions-based models

Among the utility function based models, one distinguishes piece-wise linear approaches [Siskos et al., 2005, Siskos, 1982, Figueira et al., 2008, Figueira et al., 2009], non-linear ones [Perny et al., 2016, Gilbert et al., 2017], and regret minimization-based ones [Benabbou and Perny, 2017]. These approaches mostly rely on linear programming and involve an intensive interaction with the domain expert, e.g. providing the intensity of their preferences in [Bana e Costa and Vansnick, 1999]. These preferences are used to refine the model and the process is iterated.

2.8.0.3 Choquet integral

As the Choquet Integral is a widely used model in MCDA, there has been some work dedicated to the elicitation of its parameters, the values of the fuzzy measure that parameterizes the model. A review of methods based on linear programming can be found in [Grabisch et al., 2008]. An approach was established by Benabbou [Benabbou et al., 2016b] by iteratively reducing the polytope of possible parameters until the max regret yielded by the model was bounded in a satisfying way. It was extended to several categories of MCDA problems (ranking, choosing and sorting) by the same authors [Benabbou et al., 2017]. A fixed-point based method can be found in [Goujon, 2018].

A particularity of these approaches is that they select the questions in order to ensure the best possible separation of the space, with the will to reduce uncertainty as much as possible. One of the downsides is that such questions do not allow to detect or deal with incoherences among the preferential information. Indeed, these questions are built so that the answer eliminates half of the remaining polytope, without any answer being "impossible" with regards to the previous answers.

A more recent approach based on mixed integer programming (MIP) can be found in [Beliakov and Wu, 2021]. MIP was also used for optimizing evaluations [Martin and Perny, 2020]; on the other hand, an approach aiming at reducing the domain by adding so-called *buoyancy* constraints can be found in [Beliakov and James, 2021].

2.8.1 Robustifying MCDA models

The pervasive interaction with the expert poses a risk of eventually selecting a hazardous model, particularly so in the context of combinatorial optimization: one more interaction might have shown the inconsistency of the retained model. This fact reflects the ambient uncertainty inherent to modelization [Stewart et al., 2005].

An alternative is to consider an ensemble approach, taking into account and aggregating all models consistent with the preference constraints. This was in particular tried in [Greco et al., 2005, Greco et al., 2009, Angilella et al., 2010], by sampling the region of feasible models and checking where the models disagree with each other. In particular, Stochastic Multicriteria Acceptability Analysis (SMAA) is a method designed specifically for such aggregations [Angilella et al., 2015, Saint-Hilary et al., 2017], and more specifically in ranking problems [Pelissari and Duarte, 2020]. The idea is that a preference relation a is necessarily preferred to b iff a dominates b according to all compatible models. If at least one model yields that a dominates b, but another yields the opposite, then a is *possibly* preferred to b.

Finally, some of the uncertainty on elicited model comes from the fact that the DM might give inconsistent data, or contradict themselves [Labreuche and Grabisch, 2013].

2.8.2 Limitations, and Motivations for Machine Learning

The above methods take their inspiration from the Operations Research domain, handling preferences as constraints, such as *"a is preferred to b"*, or *"value a_i is preferred to value a'_i on criterion i"*. The quality of the solution thus depends on the consistency of these constraints, and/or the interaction with the expert (supplying most informative constraints, removing constraints in case of insatisfiability). Moreover, some models are not designed for detecting inconsistencies, leading high vulnerability to even a single erroneous constraint.

The main limitation of the state of the art thus lies in the fact that the existing methods require an intensive interaction with the expert, thus being very costly in time-to-information ratio. Furthermore in the Choquet integral context, they only consider flat CIs; to our best knowledge, no approach has been developed to elicit HCI or UHCI models.

The goal of the presented work is to alleviate the above limitation, and develop Machine Learning-based approaches to learn HCI and UHCI models from data in an end-to-end fashion. While ML approaches exploit data, reporting use cases and gathered by human experts, they handle by construction (some amount of) noise and inconsistencies in these data.

The supervised ML methods used in the presented approach are presented in Chapter 3.

CHAPTER 3

SUPERVISED MACHINE LEARNING

Contents

3.1 Introduction

Machine learning [Mitchell, 1997, Amini, 2015] is a subfield of artificial intelligence. It focuses on the ability of certain systems to *learn* from data. Such a system is usually composed of a *model*, i.e. a mathematical function whose behaviour is controllable through a set of parameters; and of an *algorithm*, which is a procedure that can extract information from data in order to find the best parameters for the model to perform well on a specific task, without its behaviour being hard-coded by the developer.

Supervised Machine Learning is the specific area of machine learning where a model is given *labeled data*. This means that the model will be provided with examples whose expected output, or label, is known. The algorithm uses the data to tune the model's parameters in order to get its own output as close to the label as possible. Nonetheless, it is to be noted that the observed data is only a sampling of the examples that can be found in the real world. It is thus expected that the model not only performs well on the dataset that is used for training, but also on any other, yet-unseen example. In this case, we say that the model *generalizes* well to new data.

This process is detailed in the next sections.

3.2 Notations and General Principle

We adopt below the following notations, which will remain valid for the whole document.

- X is the data space, or the set of possible inputs of the model. It might take many forms, as long as its elements are represented as vectors; in general, we assume that X is a subset of $\mathbb{R}^n$ for a given input dimension n;
- Y is the output space, or the set of possible outputs, or labels of the model;

- $\mathcal{F}_\theta : X \to Y$ is a *predictive model*; that is, a function which assigns an output in Y to a data point in X, with θ a parameterization of the model (i.e. a set of values which define the behaviour of $\mathcal{F}_\theta$). If there is no ambiguity, we might simply write $\mathcal{F}$;
- Θ the set of all possible parameterizations (in particular $\theta \in \Theta$);
- $\text{DS} = \{(x^{(i)}, y^{(i)}),\ i \in \{1, ..., m\}\}$ is a training dataset, composed of m pairs $(x^{(i)}, y^{(i)}) \in X \times Y$, with $y^{(i)}$ the *label* of $x^{(i)}$;
- $\hat{y} = \mathcal{F}(x)$ is the prediction of the output of x by the model $\mathcal{F}$;
- $\text{err}_\theta : X \times Y \to \mathbb{R}$ is a immediate error, or *loss* function induced by $\mathcal{F}_\theta$ on an example, and $\text{Err}_\theta : 2^{(X \times Y)} \to \mathbb{R}$ is the *global loss function*, on a dataset or (in a theoretical and impossible setting) on the whole possible input space. We might also drop the subscript when not necessary.

A supervised learning model thus receives a training dataset DS, and attempts to find the θ such that the model $\mathcal{F}_\theta$ has the minimal possible error when trying to predict the label of any $x \in X$. That is, we wish to find the parameterization θ such that, for all examples $(x, y) \in \text{DS}$, we have $\mathcal{F}_\theta(x) := \hat{y}$ to be as close as possible to y. The error err is thus to be a measure of dissimilarity between $\hat{y}$ and y on all examples. Note that the error on the model depends on θ, we thus write err_θ, and omit the subscript when there is no ambiguity.

Example 20. *Assume that our data points are pictures, with resolutions of* 50 *pixels by* 50*, which represent either a cat or a dog, in three colour channels (RGB, with* 256 *possible values for each colour channel). We thus have:*

$$X = \{0, ..., 255\}^{7500},\ Y = \{cat, dog\}$$

The dataset DS *is a collection of pictures, each labeled by the animal they represent. A naive error function would be the following:*

$$\text{Err} : \text{DS} \mapsto \frac{1}{m} \sum_{(x,y) \in \text{DS}} \mathbb{1}_{(\mathcal{F}(x) \neq y)}$$

where $\mathbb{1}_A$ *has value* 1 *if* A *is true,* 0 *otherwise.* Err *thus counts the proportion of rightly classified elements among the dataset. The instantaneous error is simply* $\text{err}(x, y) = \mathbb{1}_{(\mathcal{F}(x) \neq y)}$*, i.e.* 1 *if* x *is misclassified,* 0 *otherwise.*

Formally, the overall goal is to minimize the generalization error:

$$\min_{\theta \in \Theta} \text{Err}_\theta(X \times Y) = \min_{\theta \in \Theta} \int_{(x,y) \in X \times Y} p(x, y) \text{err}_\theta(x, y) \tag{3.1}$$

With $p(x, y)$ the considered distribution on $X \times Y$.

The difficulty is that the generalization error cannot be computed in practice: we do not have access to the all possible instances x, and we do not know the corresponding y (otherwise, learning would be unneeded). We only have access to a limited training dataset, making it only feasible to compute the empirical error $\sum_{(x,y)\in \mathrm{DS}} \frac{1}{m} \mathrm{err}_\theta(x, y)$. The whole theory of statistical learning [Vapnik, 1998] aims to define *regularization terms* $R(\theta)$, such that the sum of the empirical error and $R(\theta)$ defines an upper-bound on the generalization error. The learning problem thus becomes to find a (quasi) optimum solution of the optimization problem:

$$\min_{\theta\in\Theta} \mathrm{Err}_\theta(X \times Y) = \min_{\theta\in\Theta} \sum_{(x,y)\in \mathrm{DS}} \frac{1}{m} \mathrm{err}_\theta(x, y) + R(\theta) \tag{3.2}$$

thus providing guarantees on the generalization error. The guarantees depend on the size of the training set (the larger the better) and the complexity of the model space (the simpler the better).

A key assumption of the statistical learning theory is that the dataset includes independent and identically distributed samples, and that the distribution of the training data is the same as that of the data the model will be applied to in the following.[1]

In brief, the quality of the model learned by minimizing Eq. 3.2 depends on i) the iid assumption and the fact that the training and test distributions are the same; ii) the size of the training set to be "sufficiently" large; iii) the model space to be "sufficiently simple".

Example 21. *The impact of a training set drawn after a non-representative distribution can be illustrated as following. Assume that the data from Example 20 consists only of cats, mostly on a red carpet, and of dogs, mostly on a green carpet. A model trained from these data might exploit a single feature, the color of the carpet, to discriminate cats from dogs.*

While this model behaves accurately on the training set, it will likely behave very poorly on the test set. In summary, if the dataset contains a very ***biased*** *subset of the real world space, the model will generalize poorly.*

Another issue, related with the discrepancy between the training and the test set, is referred to as concept drift*. Assume one wishes to predict the price of a house based on attributes* 1 *to* 6 *in example 1. If the dataset includes the most recent houses in the city, the learned model will tend to overprice the old houses,*

[1] The case of a discrepancy between the training and the target test data attracts growing attention under the name of *domain adaptation* [Ganin et al., 2016]; it is outside the scope of the presented work.

that usually are less expensive than new ones, everything else being equal. This bias could be prevented by gathering a representative mixture of old and new houses in the training dataset.[2]

3.3 Optimizing the Learning Criterion

A significant number of loss functions (Eq. 3.2) a.k.a. learning criteria have been defined in the machine learning literature [Bishop, 2007]. In the remainder, we limit ourselves to differentiable criteria and continuous model spaces, i.e. $\Theta \subset \mathbb{R}^p$. The search for an optimal or sufficiently good $\theta \in \Theta$, parameterizing the sought model, can thus be based on the exploitation of the gradient of the learning criterion. This section presents the principle of gradient descent and discusses its strengths and limitations.

3.3.1 Gradient Descent

A family of optimization algorithms is based on gradient descent. Considering a parameterized function $f(\theta)$, with $\theta \in \Theta \subseteq \mathbb{R}^p$, the goal is to find

$$\theta^* = \arg\min_{\theta \in \Theta} \mathcal{L}(\theta)$$

with $\mathcal{L}$ the learning criterion (the empirical error augmented with a regularization term, Eq. 3.2). The gradient descent proceeds as follows: θ_0 is initialized by drawing at random in Θ. Then, for a certain number of steps, or until convergence is reached, θ_t is updated according to:

$$\theta_{t+1} \leftarrow \theta_t - \rho \frac{\partial f(\theta_t)}{\partial \theta_t}$$

where hyper-parameter ρ is the so-called *learning* rate. A key issue consists in setting or adjusting ρ, that controls the size of each update: too small a ρ will increase the computational cost; too large a ρ will prevent the convergence of the process.

3.3.2 Strengths and limitation of gradient descent

A main strength of gradient descent is that it performs well in convex optimization problems, where the problem admits a single, global, optimum (provided that ρ is not too large, as said). In the case of non-convex problems, gradient descent tends

[2] As the model would then tend to predict some average price, an even better option is to consider a representative sample of houses, and add the age of the house as additional feature.

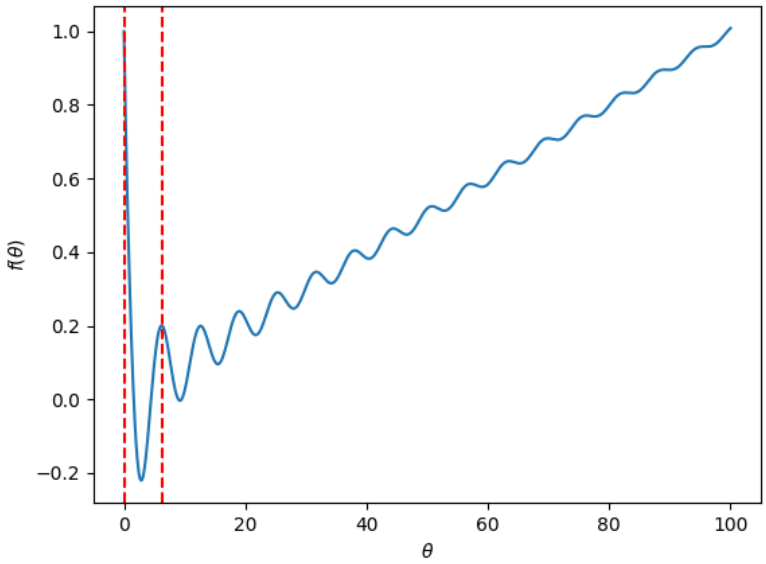


Figure 3.1: $\Theta = [0, 100]$. As f is not convex w.r.t. θ, there is a high chance that it remains stuck in a local optimum. The global optimum is the leftmost, and the vertical dotted lines define its basin of attraction. We see that there is little chance to reach this region with a random initialization of θ_0.

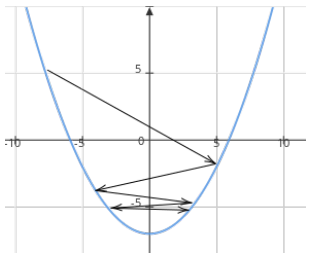
to converge to the local minimum closest to its initialization θ_0, and the eventual solution can thus be arbitrarily bad, depending on the initialization.[3]

As said, the learning rate is critical to achieve a good and fast convergence, which raises some difficulties when the norm of the gradient has a high variance. For instance (Fig. 3.1), the gradient norms are rather small for large values of θ: it will take a large number of steps to move significantly if the learning rate is small. Quite the contrary, the gradients around the global optimum are very large: if the learning rate is large, θ_t might be ejected outside of the optimum's basin of attraction.

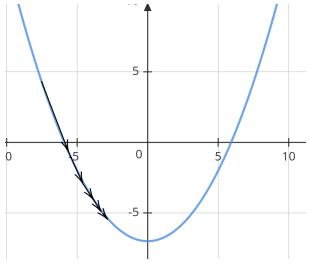
3.3.3 Algorithms Based on Gradient Descent

Several variants of the native gradient descent algorithm have been conceived over the decades. It is to be noted that, all in all, it makes no sense telling that one algorithm is "better" or "more performant" than another one. According to the

[3]Note that in the particular case of over-parameterized neural networks, that will be considered in the experiments, it is empirically noted that many local optima are actually good, i.e. yield similar performance as the global one [Choromanska et al., 2015].



(a) Large learning rate: the model bounces off the sides of the function, unable to reach a minimum



(b) Small learning rate: the model slows down too much, unable to make any significant progress after a few steps

Figure 3.2: Influence of the learning rate

No-Free-Lunch theorem [Wolpert and Macready, 1997], all optimization algorithms are equally performant when averaged over all possible problems. Nonetheless, some are more relevant for solving certain problems than other. Moreover, none of the variant presented in this section allow to get rid of the learning-rate tuning problem, although they sometimes replace it by a different parameter to tune.

A **momentum** term [Rumelhart et al., 1986] can be added to the gradient. This term conserves some information from the previous steps, in order to influence the current gradient, and keep some inertia at times where the gradient might vanish due to low absolute steepness. This might allow to cross some "hills" in the objective functions, allowing the model to extract itself from shallow local minima. It might also be useful to accelerate convergence in some configurations: it accelerates the gradient along components that are consistently present several steps in a row, while terms that contradict themselves from one step to the other can cancel each other out.

Formally:

$$\theta_{t+1} \leftarrow \theta_t - \rho \left(\sum_{i=0}^{t-1} \frac{\partial \mathrm{Err}_{\theta_i}(\mathrm{DS}))}{\partial \theta_i} m^{(t-i)} \right)$$

With m the momentum term in $[0, 1[$. That is, the previous gradients are preserved, but their influence decreases exponentially as they are further away in the past. While it provides some advantages, this algorithm thus introduces a new hyperparameter to tune, m; if m is too small, we have the same problem as in native gradient descent, while if it is too large, it might conserve obsolete information, along with increasing the size of the gradients, making the optimization unstable. It also does not dispense from the need to tune ρ.

A widely used algorithm (which is also compatible with momentum) is the **stochastic gradient descent** [Robbins and Monro, 1951]. This algorithm is

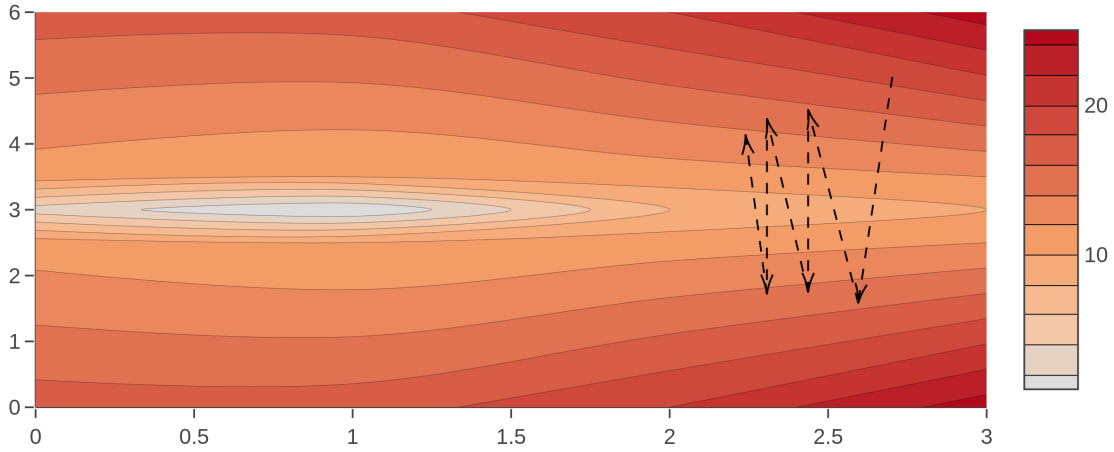


Figure 3.3: Without momentum: the model follows the local gradient only, and bounces on the walls of the valley, failing to reach the bottom

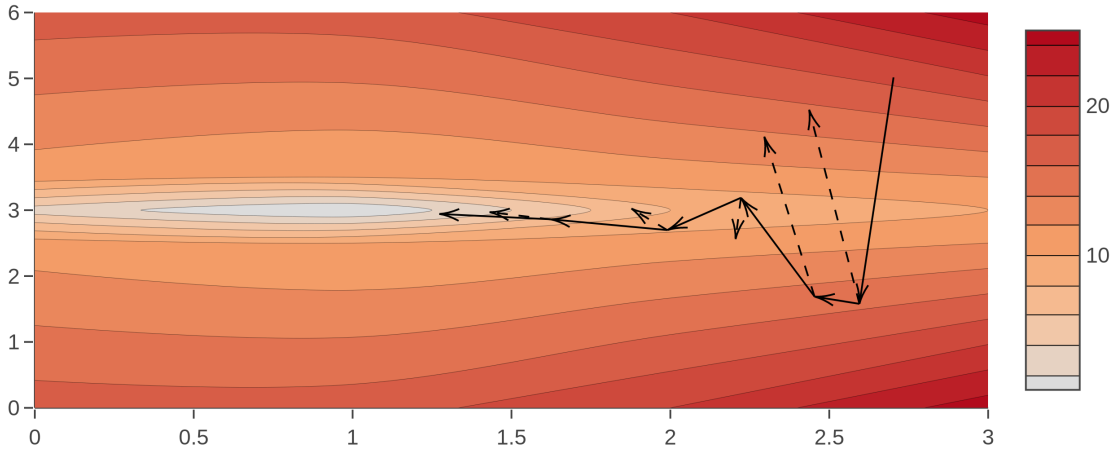


Figure 3.4: With momentum: the vertical components cancel each other out, while the decent accelerates towards the lower x values

Figure 3.5: Momentum in pathological case: the narrow valley; the optimum is in the light part (around $x = 0.75,\ y = 3$). Dotted arrows indicate the local gradient. Plain arrows indicate the step actually taken, with momentum taken into account.

based on computing the gradient of the loss with regards to a single data point rather than the whole dataset. The update step is thus written

$$\theta_{t+1} \leftarrow \theta_t - \rho \frac{\partial \text{err}_{\theta_t}(x, y)}{\partial \theta_t}$$

with a different (x, y) drawn randomly at each step. It is obvious that a stochastic step is much quicker than a usual one, as it only needs to compute the gradient for a single data point. On the other hand, each step is much less informative, some even being counter-productive, especially if the example used for that step is very noisy or biased. Nonetheless, if the dataset is large and clean enough, the large number of good examples will cancel the few bad ones. All in all, this algorithms provides a less smooth optimization, but its performances are on par or better to many other algorithms, making it widely popular in ML. An intermediate between stochastic and batch gradient descent is the **mini-batch** gradient descent, where a portion of the dataset is used for each update (more than 1 example, and fewer than the whole dataset). This allows a tradeoff between stability and efficiency.

More sophisticated approaches to adjust the learning rate along training are **Adam** [Kingma and Ba, 2015], based on estimating the moments of the gradient; **Adagrad** [Duchi et al., 2011]; **Adadelta** [Zeiler, 2012]; and **RMSProp**. The reader is referred to [Ruder, 2017] for a more exhaustive list.

3.4 Classes of Supervised Learning Problems

3.4.1 Basics

We define in this section the two main classes of supervised problems: regression and classification. In both problems, we want to infer from data the parameters θ of a model $\mathcal{F}_\theta$ that best re-attribute the label in Y to a given input in X, as expressed in Equation 3.1. Then:

- if Y is discrete, the problem is called *classification*
- if Y is continuous, the problem is called *regression*

Example 22. ***Classification:*** *In Example 20, the objective is to label new pictures by whether they represent a cat or a dog. Y is thus discrete, with two possible values, this is a classification problem. A usual loss function would be the misclassification error, i.e. the proportion of examples whose predicted label $\hat{y}$ is different from their real label y;*

Regression: *Assume the price-predicting model from Example 21. In this case, our output set of possible prices is $Y = \mathbb{R}_+$, which is continuous, making this*

a regression problem. A usual loss function would be the mean-squared error, i.e. the average squared error distance between an example's label y and the predicted value $\hat{y}$.

Another class of supervised problems is *learning to rank* [Burges et al., 2005, Burges et al., 2006, Liu, 2009]. In this case, the objective is to learn a partial order among a set of examples, from data that reflects preferences among examples (for instance, pairwise preferences, of lists of examples ranked from best to worst). A usual loss function would be the proportions of inversions in the ranking predicted by the model.

We will come down in more details on these problems in Chapter 8, when we detail the specific problems that we work on.

3.4.2 Linear Models

We describe below linear learning models, as these models are the basis for many others; their simplicity and the fact that they are relevant in many problems make them fundamental models in machine learning. Moreover, the wide use of linear models in MCDA makes the problem of learning them interesting to us; they are indeed quite easy to interpret, especially if their inputs are normalized, as is explained in Chapter 2.

Let F be the family of linear models with a one-dimensional output. That is to say

$$F = \{\mathcal{F}_w : x \mapsto wx + b \mid w \in \mathbb{R}^n, b \in \mathbb{R}\}$$

We show below two linear models, one for regression, and one for classification. In both of these cases, $X \subseteq \mathbb{R}^n$, $Y \subseteq \mathbb{R}$.

3.4.2.1 Linear Regression

The problem of linear regression consists in finding the best weight vector w so that, for any example (x, y) in the training dataset DS, $wx + b$ be as close as possible to y. It is usual that the measure of closeness is the Euclidean distance, or its square. This comes down to minimizing $\mathrm{Err}_{(w,b)}(\mathrm{DS}) = \sum_{(x,y)\in \mathrm{DS}} (wx + b - y)^2$ (least square linear regression).

Working with such models means that we assume that the output evolves linearly w.r.t. all of the values on the attributes, and that their respective influences are additive. That is, we assume that there are no interaction between attributes. Figure 3.6 illustrates this in a 1-dimensional setting.

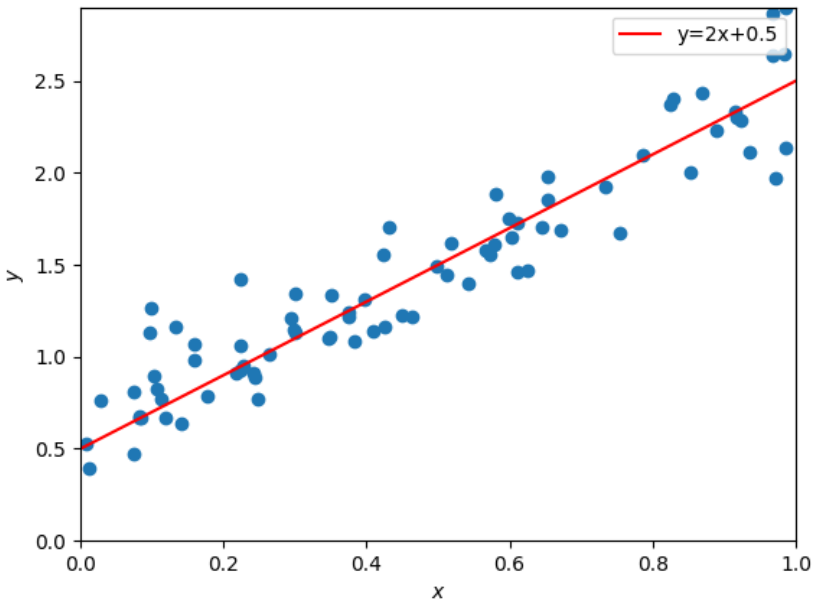


Figure 3.6: A linear regression, where the optimal w is 2, and the intercept is 0.5

3.4.2.2 Linear Classification

Assume we have two classes, 0 and 1. The aim of binary linear classification is to find a hyperplane separating the examples into their respective classes; that is, the representatives of class 0 should be on one side, the representatives of class 1 should be on the other side.

Formally, we wish to find $w \in \mathbb{R}^n, b \in \mathbb{R}$ s.t.:

$$\mathcal{F}_w(x) = \begin{cases} 0 & \text{if } wx + b < 0 \\ 1 & \text{otherwise} \end{cases}$$

A simple example of a binary linear classifier is Rosenblatt's perceptron [Rosenblatt, 1958], where the discrimination function is a Heaviside step function in b.

A softer model is logistic regression [Cox, 1958], which can be seen as both a classification and a regression algorithm. The *regression* in the name comes from the fact that this algorithm aims at determining the probability of an input belonging to class 1 rather than class 0. It can thus be used as a linear classifier with a confidence index appended to the result-class.

The assumption is that, for any input x:

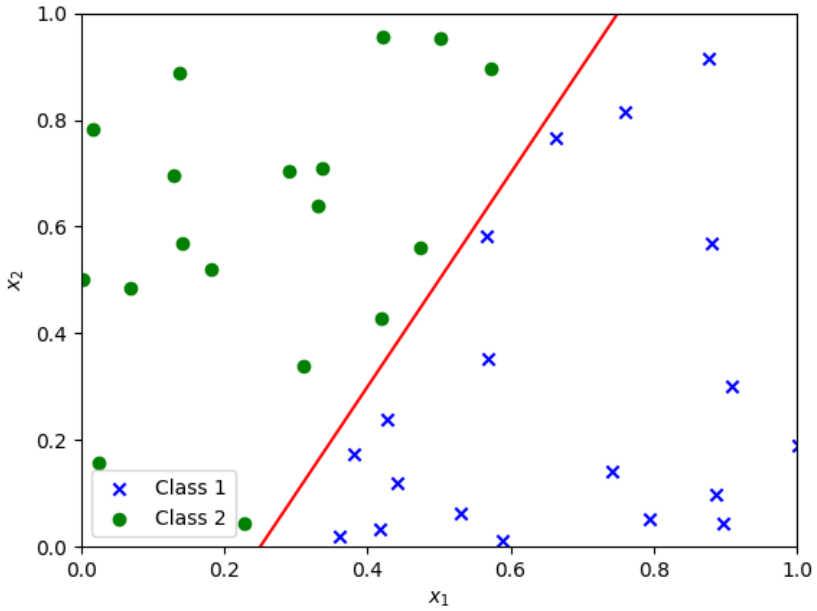


Figure 3.7: Binary 2-dimensionnal classification: the red border (hyperplane) separates both classes well

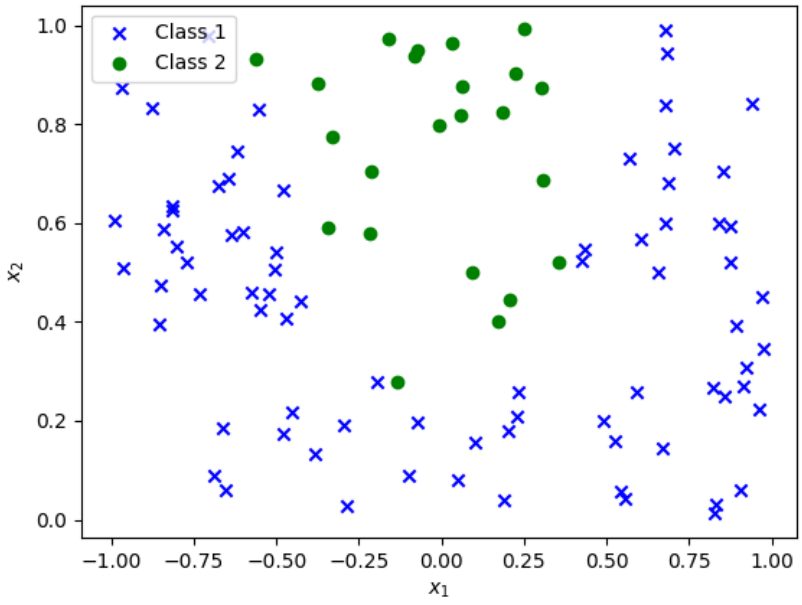


Figure 3.8: Binary 2-dimensionnal classification: the classes can not be linearly separated

$$log\left(\frac{\pi_x}{1-\pi_x}\right) = \beta + wx \tag{3.3}$$

where β is a bias term, π_x is the probability of x belonging to class 1 and $w \in \mathbb{R}^n$ is a weight vector. Basically: the log of the ratio between the probabilities of x belonging to either class is a weighted sum on x. Logistic regression will then learn w and β.

This permits to write π_x as:

$$\pi_x = \frac{1}{1 + e^{-(\beta+wx)}} \tag{3.4}$$

The aim of the methods is then to learn the optimal w and β in order to fit the data as much as possible. It can be noticed that the "indecision area", or the set $\{x \in X \mid \pi_x = 0.5\}$ is again an hyperplane. This model can thus be used as a linear classifier, by assigning to each point the label of the half-space it belongs to.

When there are more than two classes involved, basic methods involve learning several linear classifiers, and then aggregating their outputs, or votes, in order to decide on the class. A usual way of doing this is to train one classifier for each class, such that the classifier separates the space between the elements that belong to that class, and those that do not [Amini, 2015]. This method is called *One vs All*.

Note that other linear-based models exist, such as support vector machines (SVM) [Boser et al., 1992]. The main idea behind SVMs is to warp the data into a space of higher dimension, where it becomes linearly separable with a wide margin (i.e. the separating hyperplane tries to remain "as far as possible" from the datapoints). This idea of heavily warping the input space in a way that allows for easy separation, or regression, is also at the root of neural networks, a class of models we now present.

3.5 A Universal Approximator: the Neural Network

Linear models are of strong interest, and they are still widely used today. Nonetheless, they have an obviously limited representation capability. That is, in real life applications, a linear model is often not enough for solving a given problem. In the presented work, we focus on neural networks, because of their flexibility and the fact, as will be shown in Part IV, that we can design a neural architecture corresponding to our target model space, i.e. UHCIs (defined in Chapter 2).

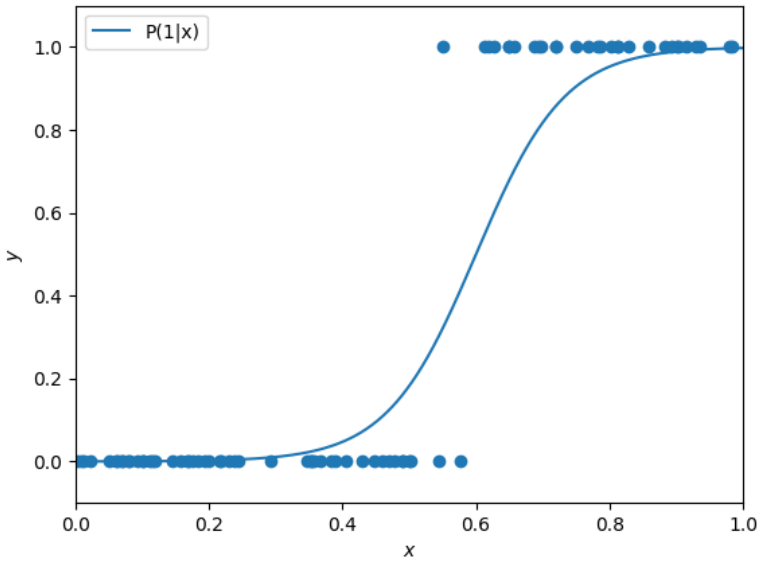


Figure 3.9: Logistic Regression on 1-dimensionnal inputs: showing the probability π_x given x. The dots are data points, with theit y value being their observed class.

3.5.1 Neurons

A *neuron* is the base unit of a neural network. A neuron is usually a multi-input function, which aggregates its inputs into a single output via a non-normalized weighted sum, then returns the image of this aggregation by a non-linear function called its *activation*. Formally, let ϕ be a neuron with n inputs, a weight vector $w = (w_1, ..., w_n)$, a bias $b \in \mathbb{R}$ and an activation function $a : \mathbb{R} \to \mathbb{R}$. Given an input $x = (x_1, ..., x_n)$, we have:

$$\phi(x) = a\left(\sum_{i=1}^{n} w_i x_i + b\right) = a(wx + b)$$

Each neuron is thus in itself a small, non-linear operator, which offers some representation power through the possible modification of its parameters (bias and weights). Usual activation functions include:

- Logistic Sigmoids: $\sigma : x \mapsto \frac{1}{1+\exp(-x)}$;
- Inverse Tangent Function;
- Rectified Linear Units: $\text{ReLU} : x \mapsto \max(x, 0)$;

- Leaky ReLU: LReLU : $x \mapsto \max(\alpha x, x)$, with $\alpha \in]0, 1[$;
- Softplus: SoftPlus : $x \mapsto \ln(1 + \exp(x))$.

Of course, activation functions are not limited to these classes. In fact, there is little formal restriction as to what should be used as an activation function. Nonetheless, some properties are highly suitable for a:

- continuous, as discontinuities might make training highly unstable;
- easily differentiable almost everywhere, such that a gradient or subgradient might be quickly computed for gradient descent;
- non-linear, otherwise the neuron itself is just a linear function of a linear function, making the activation useless and limiting the expressivity;
- non-saturating (i.e. with a gradient that tends towards 0 on either side of $\mathbb{R}$), as that might cause the gradients to quickly become too small for the model to learn (vanishing gradient phenomenon).

The last point is one of the reason as to why sigmoids have lost popularity in the past years, to the benefit of ReLU and its variants: a composition of several successive sigmoids tended to kill off the gradients. A more comprehensive analysis of activation functions can be found in [Nwankpa et al., 2018]. Note that a single neuron with a Heaviside activation is a perceptron as described above.

Below, for simplicity reasons, we write indifferently h_i both for a neuron or the value output by this neuron, when no ambiguity arises.

3.5.2 Feedforward Neural Networks

Considering a neuron as described above, we can now create a so-called "layer" H. Assume m neurons $h_1, ..., h_m$, each with n inputs. Assume a bias vector B_h in $\mathbb{R}^m$, and a weight matrix $W_h \in \mathbb{R}^{m \times n}$. We also assume that neuron h_i has an activation function $a_i : \mathbb{R} \to \mathbb{R}$. Then, given an input vector $x = (x_1, ..., x_n)$, we compute the output of h as:

$$\begin{aligned} H : \mathbb{R}^n &\to \mathbb{R}^m \\ x &\mapsto (h_1(W^{(1)}x + b_1), ..., h_m(W^{(m)}x + b_m)) \end{aligned} \tag{3.5}$$

with $W^{(i)}$ the ith line of W.

Now, we assume that we have a number L of layers H_1 through H_L, such that, for all $j \in \{2, ..., L\}$, the dimension of the input (or dimension) of H_j is equal to

the number of neurons (or width) of H_{j-1}. We also write n the width of H_1, and m the width of H_L then we can compose those functions, and compute:

$$\begin{aligned} \text{FNN} : \mathbb{R}^n &\to \mathbb{R}^m \\ x &\mapsto H_L(H_{L-1}(...H_1(x)...)) \\ &= A_L(W_L A_{L-1}(W_{L-1}...A_1(W_1(x))...)) \end{aligned} \tag{3.6}$$

with A_j the function which applies the activation function element-wise to its input; that is, writing m_j the width of H_j ($m_0 = n$ for simplicity):

$$A_j(x_j) = (a_1(x_1), ..., a_{m_j}(x_{m_j}))$$

FNN is called a *Feedforward Neural Network*, or *Multilayer Perceptron*. H_1 is called the *input layer*, and H_L is called the *output layer*. If we call X the input space, and Y the output space, then FNN is a function from X to Y, parameterized by:

- a set of weight matrices W_i for i in $\{1, ..., L\}$; we have $W_i \in \mathbb{R}^{m_i \times m_{i-1}}$, and $B_i \in \mathbb{R}^{m_i}$.
- a set of biases $\{B_1, ..., B_L\}$

It is the simplest architecture of neural networks, yet has a very high representation power. In fact, according to the **Universal Approximation Theorem** [Hornik et al., 1989], a single-layer FNN (given enough width for the said layer) is enough to approximate any Borel-measurable function. In our example, we can count the number of free parameters to understand that flexibility:

- $\sum_{i=1}^{L} m_i m_{i-1}$ weights (summing the number of elements in all matrices)
- $\sum_{i=1}^{L} m_i$ biases

For wide and deep neural networks (deep means that L is large), the number of parameters augments at a polynomial rate of the width of each layer. Figure 3.10 represents a FNN with 1 hidden layer.

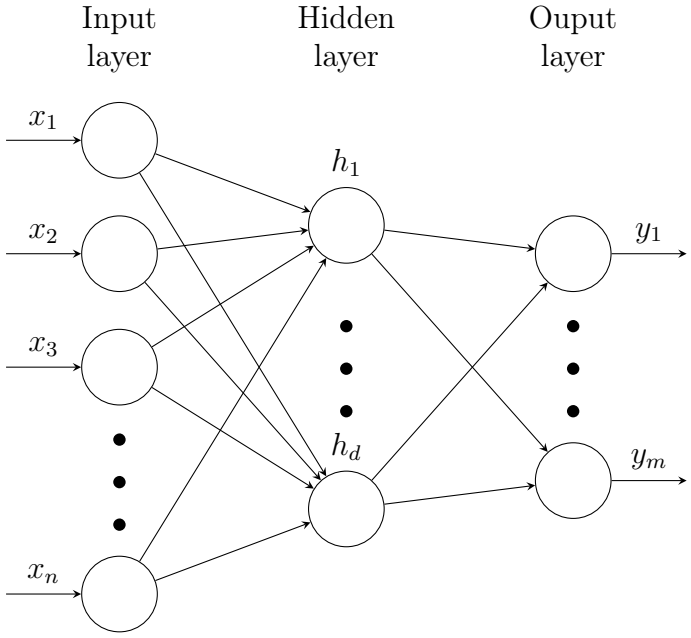


Figure 3.10: A multilayer perception with a single hidden layer, an n-dimensionnal input, an m-dimensionnal output, and d hidden neurons on its hidden layer. An arrow from neuron a to neuron b means that the output of a is an input of b.

3.5.2.1 Training an FNN

An FNN is usually trained using the **backpropagation algorithm**. This algorithm is basically the chain rule applied to the expression in (3.6). The idea is to compute the gradient of the error w.r.t. the output, then backpropagate, layer by layer, until the gradient of the error with regards to all of the parameters has been computed. Then, all of the parameters are updated at the same time.

The parameters upon which one can act are the weights and biases. In order to compute the gradients of the parameters of layer H_j, one must first compute the error Err(DS), along with its gradient w.r.t. the output layer H_L. For instance, for the kth neuron of H_L, noted h_k^L, we have:

$$\delta_k^L := \frac{\partial \text{Err(DS)}}{\partial h_k^L}$$

Doing that for all neurons in H_L, we have a gradient vector $\delta^L = (\delta_1^L, ..., \delta_{m_L}^L)$. Then, for all $j \in \{1, ..., L\}$, we have:

$$\begin{aligned}\delta^j &= \frac{\partial \text{Err}}{\partial H_j} \\ &= \frac{\partial \text{Err}}{\partial H_{j+1}} \frac{\partial H_{j+1}}{\partial H_j} \\ &= \delta^{j+1} \frac{\partial H_{j+1}}{\partial H_j}\end{aligned} \tag{3.7}$$

We see from Equation (3.5) that the gradient $\frac{\partial H_{j+1}}{\partial H_j}$ is easy to compute. Indeed, we have:

$$H_{j+1} = A_j(W_j H_j + B_j)$$

Thus

$$\frac{\partial H_{j+1}}{\partial H_j} = \nabla A_j(W_j H_j + B_j) W_j$$

as the activation has been chosen easy to differentiate, this gradient is quick to compute. By applying this on all layers, we can efficiently compute the gradients for both the weights and biases, thus allowing for gradient descent.

3.5.3 Other Architectures

It is to be noted that neural networks are not resticted to FNN, nor to linear neurons, and that their applications extend beyond supervised learning. In fact, in [Lecun et al., 1998], the authors describe neural networks in a much more general way, as interconnected modules that can be optimized through gradient descent.

Many more architectures exists, for different problems. In particular:

- Recurrent Neural Networks [Hochreiter and Schmidhuber, 1997]
- Convolutional Neural Networks [Khan et al., 2019], which are widely used in image processing, or when there exist spatial correlations among the data's dimensions
- In unsupervised or semi-supervised learning, autoencoders and their variational counterparts [Kingma and Welling, 2014] are widely used, along with energy-based models built upon neural decoders [Song and Kingma, 2021]

More architectures can be found in [Liu et al., 2017]. The ability of a neural network to approximate any function is its main strength. Nonetheless, it also brings about several drawbacks, as we see in the following section.

3.6 Classic Difficulties in Machine Learning

3.6.1 Over- and Under-fitting

We present here the notions of underfitting and overfitting, two key concepts in machine learning and interpolation in general.

Underfitting is the phenomenon that happens when the chosen model is too rigid (too many constraints), or too simple (too few parameters), to accurately represent the underlying structure of the data. As a consequence, the model is not able to fit the data properly, and is forced to oversimplify its representation of the data, resulting in the loss of a non-negligeable part of the information contained in the training set. For instance, a linear model will easily underfit complex problems, which might require more complex structures.

Overfitting is the opposite phenomenon. When the model chosen has too much freedom to fit the data, it might end up considering noise, or random artifacts, as information contained in the data it can see. It will then learn too much from the training data. This will result in good performances on the specific examples from the training set, but a bad generalization ability, thus bad performances on real life applications. This is often the case for small datasets, where the noise is not compensated by the number of examples.

In order to counter overfitting, there are several widely used strategies. The most popular ones are:

Using a simpler model: downgrading the representation power of the model used can be done by using a subclass (for instance, linear instead of quadratic), or by removing parameters (for instance, removing layers in a neural network). This restricts the class of models that can be learned, leading to more constrained behaviours.

Regularizing the model: regularizations are terms that can be added to a loss function in order to penalize some behaviours. For instance, if we wish to learn a linear model $\mathcal{F}_w$ for regressing, we might want to encourage the model to drop as many weights as possible (that is, to take into account as few attributes as possible, setting the other weights to 0).

If we use the so-called *lasso*, we use the L1-norm of w to penalize the loss function Err, leading to a new error function Err^* to minimize:

$$\text{Err}^*(\text{DS}) = \text{Err}(\text{DS}) + \lambda \sum_{i=1}^{n} |w_i|$$

The summing term to the right leads the model to reduce its weights as much as possible. λ is a positive term which gives the extent to which the model is regularized. Note that a large λ would lead to a highly penalized model, and thus eventually to underfitting. It is thus a hyperparameter to be tuned. Note that a regularization could be any numerical function that penalizes an unwanted behaviour; frequent examples include, but are not limited to the L2 norm of the parameters or Kullback-Leibler divergences between distributions.

Dropout: very large models, such as neural networks, are called *overparameterized*, because they have an excess of expressivity w.r.t. the functions that they have to learn. Some methods were developped to prune some neurons [Le Cun et al., 1989]. On the other hand **Dropout** [Srivastava et al., 2014] is one that aims at forcing redundancy among the information learned by different neurons. The idea is that, at each training step, a randomly selected subset of neurons are deactivated (i.e. their output is set to 0), and reactivated afterwatards. As the set of usable neurons change at each step, this means that the model uses effectively a smaller number of parameters at each time than what is actually available in the model. This method, repeated over several training steps allows to force neurons to duplicate information, such that, at a given step, the used neurons compensate for the deactivated subset.

Underfitting and overfitting are illustrated in Figure 3.11. In terms of the bias-variance trade-off, underfitting shows high bias and low variance, while overfitting shows high variance and low bias.

3.6.2 Testing, and Validation Sets

Now that we have seen how to train a given model, it is crucial to be able to test it in order to evaluate its performance on future, yet unseen, inputs. Indeed, the model could learn to fit perfectly the training dataset without being able to generalize to new examples. A very widespread method, which we use here, is called **cross validation**. The main idea is simple: the performance of the model must be evaluated on data which was not used for training (i.e. data that the model has never seen before).

The process is the following: the dataset DS is split between two sub-datasets:

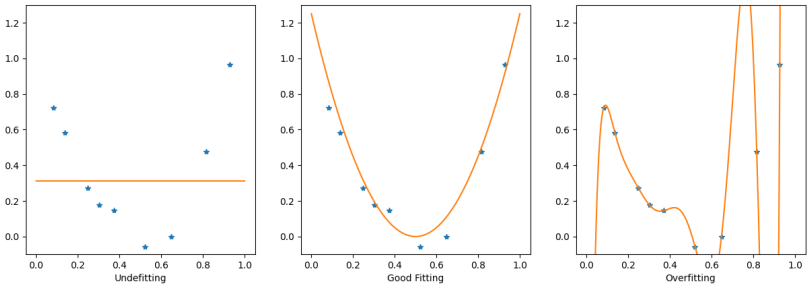


Figure 3.11: Fitting the data with different complexities, we try to regress $y = f(x)$ with f being: Linear (left), Quadratic (middle), High-order polynomial interpolation (right)

- a *training dataset* DS_{Tr}, on which the model will be trained
- a *testing dataset* DS_{Te}, on which the model will be evaluated

such that $\langle \text{DS}_{Tr}, \text{DS}_{Te} \rangle$ for a partition of the initial dataset.

The model is then trained on DS_{Tr}, and the error computed on DS_{Te}. Thus, an error that is much lower on DS_{Tr} than on DS_{Te} indicates overfitting. On the other hand, similar errors on both datasets might show that the model is able to generalise well. The evaluation of the model's performance must be done on DS_{Te} only, which is also supposed to be representative of the real life distribution of examples.

Nonetheless, it would often be useful to have an idea of the performance of the model on unseen data during training, in order to, either, stop the training before the model starts overfitting (a method known as "early stopping"), or to compute criteria for adapting the hyperparameters in real time. Obviously, this cannot be done on DS_{Te}, as that would mean biasing the model to fit the testing set, thus invalidating the performance on the latter. The usual method is thus to introduce a third set, called a *validation set* DS_V, which is a sub-part of the training set.

DS_V is "seen" by the model during training, but is not used for computing any gradient. It is only used for the training algorithm to evaluate performance on unseen data, and thus stop, or tune its hyperparameters accordingly. Nonetheless, when evaluating a model, DS_{Te} only should be used, as it was not seen at all during training.

The main tradeoff on the size of the three datasets is that:

- increasing the size of DS_{Tr} permits to use more data for training, thus improving the training
- increasing the size of DS_{Te} permits for higher accuracy on the estimation of the performance on unseen data

- increasing the size of DS_V permits for higher confidence in early stopping and improves training

Obviously, the larger one set is, the more we need to reduce at least one of the others. A very large training set with a very small validation and testing set leads to the model obtaining all of the available data for training, but very little to prevent overfitting. On the other hand, a very large testing set will yield a good estimate of the performance of the model, but the latter is trained on little data, and thus probably not that good. These sizes have to be chosen according to the size and noisiness of the dataset.

Moreover, it is usual to train and evaluate a model several time on different partitions $\langle \mathrm{DS}_{Tr}, \mathrm{DS}_V, \mathrm{DS}_{Te} \rangle$; this allows to remove some bias in the evaluation, as the performance is now given as the mean and variance of the error on several splits. In particular , k-fold cross-validation is a popular setting where DS is split into k equal parts; then, the model is trained on $k-1$ parts, and tested on the last part. The model is thus trained k times, switching the part used for testing each time.

3.7 Machine Learning for Safety-Critical Contexts

3.7.1 Uncertainty in Machine Learning

Machine learning algorithms aim at eliciting the parameters of chosen model. They do so by extracting information from data that is usually pulled from observation of the real world (e.g. sensor values, websites metadata, products characteristics...). We focus here on FNNs.

These models, as they are trained on data, have a highly probabilistic nature. By that, we mean that there is often some uncertainty on the result, both on the parameters learned, and (as a consequence), on the output (and thus the performance). We describe the different sources of uncertainty below.

- Uncertainty inherent to the data
  - The data that composes the datasets are observations of real world situation. As no sensor/predictor is perfectly accurate, there is necessarily noise on at least some of the values recorded.
  - some phenomena are very complex, as they have to take into account more attributes than what the data provides. As a consequence, there might be information that is missing in order to accurately and fully determine the model. The second part of Example 21 illustrates this; indeed, the age of the house is unavailable to the model, forcing it

to yield a predicted averaged value, which might be wrong if the said attribute is, in fact, very correlated to the output.

  - The data is a finite sampling of all possible observations. In particular, especially in large dimensional input spaces, there will necessarily be sparse areas, where there is little to no information as to how the model should behave there.
  - There might be biases in the data, and it will in any way never be distributed exactly as in real life; as a consequence, even though the algorithm performs well on a given dataset, it might not generalize to real life applications.

- Uncertainty inherent to the model
  - The model might not be adapted to representing the data well. In particular, might overfit or underfit
  - In some cases, especially the very popular deep learning setting, where networks usually have a large number of layers and neurons (and thus parameters), the model might be too complex for any formal guarantee to be obtained. This problem is especially true in areas from where few data points were sampled, but it might also happen very close to actual training datapoints, especially in high dimensions (adversarial examples).
  - Depending on the optimization algorithm and of the loss function chosen (e.g. stochastic gradient descent, minibatch), there might be uncertainty on the point where the algorithm stopped. It might be a local minimum, or simply a point where the gradients were too small to actually make any progress towards a global minimum. There are thus no guarantees that we effectively minimized the loss function.

All in all, the trained model is in itself a random variable (as it is highly dependant on the training sampling and other sources of uncertainty. This makes machine learning methods a priori not adapted to safety-critical environments and applications. Indeed, these systems require guarantees on the decisions they take, as a single mistake can prove disastrous. Such systems are, for instance, self driving cars, autonomous systems in nuclear plants, or air traffic management systems.

As a consequence, there has been some literature, on trying to adapt the ability of machine learning to automatically extract information from data to such systems where control over many important properties are desirable/required. High-level motivations and leads can be found in [Varshney and Alemzadeh, 2016].

We focus here on classification methods, learned from data that might be noisy or faulty.

3.7.2 Taking Uncertainty into Account

Counteracting the effect of such uncertainty is a studied problem in statistical learning. One approach is to better quantify the said uncertainty [Abdar et al., 2020], in order to evaluate the trust one can have in a model's output.

A widely used method to robustify a machine learning predictor is to evaluate an example on several models that are equally likely [Machida, 2019]. Then, one removes the subjectivity that intervenes when a single model is chosen. Moreover, this method assumes that, while one model can have weaknesses, and as such misclassify or mislabel one example, the others will "compensate". One issue with this approach is that, if this reasoning is probabilistically sound, it would require the models to have independant probabilities to fail on a specific example. Considering that the data on which they were trained are all the same, it is nonetheless likely that they will have similar bias and fail on the same examples. This is a direct application of so-called *ensemble methods* [Zhou, 2012], where multiple models are used to robustify of improve a ML system.

Jiang et al. [Jiang et al., 2018] proposed a method to verify whether a classifier can be trusted by analyzing, a posteriori, the distance between the closest class and the second closest, in order to derive confidence metrics.

A popular approach is the dual representation of uncertainty as part epistemic, part aleatory. Epistemic uncertainty is that which comes from a lack of knowledge, and is thus reducible, while aleatory uncertainty comes from intrinsic randomness of the world, and is thus irreducible. Classifiers taking this framework into account can be found in [Senge et al., 2014, Nguyen et al., 2018]. *Active learning* is a field where the training examples are chosen actively, through queries, during the training in order for the learning phase to be as quick as possible, by avoiding redundancy, for instance. Nguyen in [Nguyen et al., 2019] propose an active learning approach that aims at asking a decision maker to label the examples on the ouputs of which there is the greatest epistemic uncertainty, thus effectively aiming the uncertainty reduction at the reducible part.

Being able to evaluate the uncertainty associated with a given model's output is interesting from the point of view of robustness. Indeed, it allows for the user to know when to trust a given model, or even for the model to "decide" not to yield any output when uncertainty is too high.

3.7.3 Formal Properties for Learning Systems

The approaches that aim at quantifying and reducing uncertainty might not prove enough for certain applications. In particular, some applications require formal guarantees on the model, which are usually harder to obtain on machine learning algorithms than on classical hard-coded algorithms, mainly because:

- There are usually a large number of parameters for a given model
- The parameters are learned from data that is drawn from a certain unknown distribution
- There might be large amounts of data

The objective of these methods is to formally prove that the relation between input and output guarantees some boolean properties (SAT problems) or more complex ones which can be reduced to SAT problems (SMT problems).

Such approaches (and others) are surveyed and proposed in [Bunel et al., 2017b, Bunel et al., 2017a, Ehlers, 2017]. ReLUplex adapts the simplex algorithm to neural networks with ReLU activations [Katz et al., 2017]; the properties proven involve, for instance, *"If two drones are sufficiently far enough, the network will not consider that they are at a risk of collision"*. Other types of activations are studied, as sigmoids [Pulina and Tacchella, 2010]. These approaches use mixed integer programming, as well as overestimation of the estimation functions [Singh et al., 2019].

3.7.4 Adversarial Examples

Adversarial examples [Szegedy et al., 2013] are a recurrent problem in complex learning systems. In a classification problem, a data point is considered adversarial if it is indistinguishably close to another point (for a human being), yet is not assigned the same label by the model.

Formally, let $X \subseteq \mathbb{R}^n$ be the input space of our problem, and Y be the labels set. Let $\delta \in \mathbb{R}^+$. Finally, let $y_x \in Y$ be the label assigned to input x by the model. Now, let $(x, \epsilon) \in X \times \mathbb{R}^n$ s.t.:

$$\|\epsilon\| < \delta$$
$$y_x \neq y_{x+\epsilon}$$

Parameter δ is chosen so that the difference is too small to be detected by a human, or at least too small to change the label a user would assign to the input. Nonetheless, the perturbation ϵ still manages to fool the classifier, and ϵ is called an *adversarial perturbation*. The data point $x + \epsilon$ is called an adversarial example. They usually appear in high dimensions [Goodfellow et al., 2014], as they exploit the flexibility of complex models, such as deep neural networks. In contexts such as the ones in which Thales has applications (e.g. safety, security, defense), it might be the case that the models be submitted to adversarial attacks (i.e. adversarial examples generated intentionally to fool the model) by an antagonistic system; it

is thus important to present some guarantee against such attacks if the models are to be trusted.

Formal verifications of robustness to adversarial examples is a recent, but widely studied topic. Research includes methods to assess the robustness of black-box trained neural networks through different metrics [Bastani et al., 2016, Wang et al., 2018] or by identifying safe regions [Huang et al., 2016, Gopinath et al., 2017].

While some approaches aim at building adversarially-robust neural networks [Madry et al., 2018], some consider them unavoidable consequences of using unconstrained neural networks [Ilyas et al., 2019].

Nonetheless, in the case of constrained models, a popular approach is the use of Lipschitz-network [Hein and Andriushchenko, 2017], where the gradient of the model is bounded. This means that a small disruption in the input can only provoke a limited modification of the output, thus effectively counteracting, by design, the adversarial effect. While such models are harder to train due to the added constraints, they were shown to suffer little to no loss of expressivity [Bethune et al., 2021]. Nonetheless, this constraint is only efficient in networks with a small dimension; indeed, a small disturbance on a large number of dimensions might lead to a large global disturbance, thus allowing for adversarial examples.

We have seen that ML models are highly vulnerable to the uncertainty inherent to the data. As presented, some approaches have been developed to improve the robustness of the learned models. But what is needed here is some guarantees in a non-probabilistic sense (as provided by the statistical learning theory), about the error in particular cases. We have seen in Chapter 2 that some models stemming from MCDA and MAUT offer constrainedness and interpretability. This, in turn, allows assessment of the model by an expert, and permits the use of the model in safety-critical settings. We now present the field of *preference learning*, in particular the sub-field that focuses on learning MCDA models. Indeed, our main technical contributions will be inscribed in the wake of the methods that will now be introduced, aiming to learn, from data, models such as the previously presented UHCIs.

3.8 Preference learning

Preference learning is the subfield of machine learning that aims at learning preference and ordering relations. It is a different field than MCDA, in particular in the fact that there is little to no interaction with the DM in preference learning. In particular, there might not be the same need for transparency and interpretability of the models.

3.8.1 Preference Learning Tasks

We present here three main categories of problems adressed by preference learning, taken directly from [Fürnkranz and Hüllermeier, 2011].

Label Ranking: Label Ranking can be seen as a generalization of classification, where, instead of returing a single class $y \in Y := \{1, ..., K\}$, the model returns a preference ordering on the labels (i.e. a permutation of Y). In essence, the model, given an example x, learns a preference relation $\succ_x$ on the labels. Then, given $y, y' \in Y$, $y \succ_x y'$ means that label y is prefered to y' for classifying x.

Example 23. *A movie streaming plaftorm knows some data about its users. They also propose four types of movies:*

$$Y = \{Action, Drama, Comedy, Romantic\}$$

Given a new user profile x, the platform might want to recommend movies that the customer likes. If we have:

$$Comedy \succ_x Action \succ_x Drama \succ_x Romantic$$

then the platform can recommend mostly comedies and action movies to the user.

Instance Ranking: In this setting, the model is given a training set of examples from X, along with time their corresponding class $y \in \{1, ..., K\}$. We assume that there is a preference order on the elements of y, such that an example labeled with class i is strictly preferred to any in class $i - 1$. Then the model aims at learning a preference relation $\succ$ among the elements of X; that is, given any set of elements of X, the model can re-order them from least to most preferred.

Example 24. *A movie streaming platform classifies its movies among three categories:*

1. *those that were upvoted, or liked, by fewer than 60% of the watchers*
2. *those that were upvoted, or liked, by more than 60%, but fewer than 90% of the watchers*
3. *those that were upvoted, or liked, by more than 90% of the watchers*

Then, given the metadata of all the movies from the next batch to be added to the catalog, the model can sort those movies by a prediction of how much they

will be liked by the users. This allows the platform to focus its advertisement and suggestions on those of the movies that are more likely to be appreciated by the users.

Object Ranking: The aim of this setting, like Instance Ranking, is to learn to sort the examples by preference. Nonetheless, this time, the incoming data is not labeled by a class; instead, the training data consists of data that is already ordered from least to most preferred (i.e. the label is implicitly given by the position of an example in the ordering).

Example 25. *The same movie streaming platform ranks all of its movies from least to most liked, and uses this to train the model, without even indicating the percentage of upvotes. Once trained, it can order new data, just like in Instance Ranking.*

An early approach by Cohen et al [Cohen et al., 1999] proposes a method to aggregate on-line the preferences of several weighted experts. Learning from sets of pairwise comparisons between options has been investigated by Fürnkranz and Hüllermeier, with the use of several machine learning methods, in particular classification [Fürnkranz and Hüllermeier, 2003, Hüllermeier et al., 2008].

Active learning approaches were tried in contexts where there was some feedback (possibly incomplete, or less informative than what having the DM at hand could provide). Such approaches can be found in [Qian et al., 2015, Bärmann et al., 2017, Bourdache and Perny, 2019].

3.8.2 Multi-criteria Preference Learning

As we have seen, uncertainty is unavoidable in most learning contexts. It was thus considered that, as it was dangerous to use a learned model on its own to accomplish safety-critical decisions, one could apply the MCDA philosophy (i.e. build models for aiding a human, rather than for taking a decision). The idea is that, if we can learn an MCDA model from data, we can obtain a model that is interpretable, formally constrained, and verifiable by design; that is, once the model is trained, there is no difference at all between using it and using a "traditionally"-elicited model. The-thus verified model is thus totally free of the issues induced by uncertainty in the training data, as is one that was built with the help of a DM.

Moreover, the model is built at a fraction of the cost of building it through the usual MCDA methods of interviewing an expert, as one can use data that is already

here (for instance, logs from previous decisions). We present here the approaches developed to elicit MCDA models through learning. This field is sometimes called Multi-Criteria Preference Learning (MCPL).

A probabilistic approach for eliciting the utility functions of a AU model was implemented by [Bous and Pirlot, 2013].

A first method for learning a fuzzy measure was proposed by Grabisch [Grabisch, 1995]. This approach is a supervised regression using gradient descent, under the constraints that befit a fuzzy measure. A modification to Grabisch's algorithm was implemented in [Alavi et al., 2009] and achieves better results. This paper also tries a genetic algorithm method to learn fuzzy measures, but concludes that it is less efficient than the modified gradient descent. A more recent method involve linear programming based on data [Beliakov and Wu, 2019].

A great deal of work was accomplished by Tehrani et al on learning binary classifiers based on the Choquet integral [Fallah Tehrani et al., 2011, Fallah Tehrani et al., 2012, Hüllermeier and Tehrani, 2013]. These approach generalize the logistic regression by replacing the linear part (weighted sum) by a Choquet integral. An even more general model, called Choquistic Utilitaristic Regression [Fallah Tehrani et al., 2014] was also learned. This model is a Choquistic regression, where the utilities that rescale the parameters are learned at the same time as the fuzzy measure of the aggregator. More recently, Havens et al. [Havens and Anderson, 2018] extended these methods to non-monotonic fuzzy measures. Finally, Bourdache [Bourdache et al., 2019] recently used Bayesian linear regression to regress the parameters of weighted models, such as 2-additive Choquet integrals or ordered weighted averages.

They thus try to tackle the problem of, given data, a decision maker that can generate more data, and a model family, elicit preferences. The uncertainty there comes from the fact that the data does not reduce the set of compatible parameterizations to a single point; as a consequence picking only one single parameterization among the feasible region is highly subjective. The ideas developped below try to look at all possible models, and elicit preferences according to those. These approaches were developed around MCDA, and thus use mainly piecewise linear models.

Active learning is a learning procedure which, rather than learning from a fixed dataset, uses an oracle (expert or program) which can label examples as the learning phase progresses. This allows the learning system to request the most informative examples. This approach was used to obtain a set of parameterizations for linear models in order to obtain a set of models which minimizes the maximal min-max regret [Benabbou et al., 2016b]. In [Benabbou and Perny, 2017], the authors assume that the output is not deterministic w.r.t. the input, but they assume knowledge about the conditional distribution of the output knowing the

input, and they apply the same active learning procedure.

Approaches for learning hierarchical interpretable models include, notably, [Senge and Hüllermeier, 2011], where the trees learned are binary, but can have the same attribute intervening several times. In the case of 2-level hierarchical models (not necessarily trees), an approach exploiting such models for data fusion can be found in [Beliakov et al., 2021]; it offers elements of simplification for easing the learning of such models, when using Choquet integrals.

3.8.3 Dealing with Uncertainty

Preference learning is still machine learning, and is as such also subject to the uncertainty evoked in the previous sections. Some of the uncertainty on elicited model comes from the fact that the DM is not perfect, and might give inconsistent data, or contradict themselves [Labreuche and Grabisch, 2013].

Cailloux and Destercke present in [Cailloux and Destercke, 2017] reasons as to why the modelling of preferences should be incomplete, meaning that there must be some cases for which a model cannot determine which of two alternatives is the better one, but instead conclude on an impossibility to choose.

Theories such as possibility theory [Dubois and Prade, 2015] and belief theory [Shafer, 1976] have models to describe uncertainty with more information than a single number (as probability theory does), allowing deeper analysis.

Greco et al proposed the robust ordinal regression [Greco et al., 2009], later extended to the Choquet integral [Angilella et al., 2010] as a way of establishing a reliable decision-making tool. The idea is, given a class of parameterized decision models (with a piecewise-linear relationship between input and output), to elicit through linear programming the set of all possible parameterizations. Then, it is possible to compute, for a given decision, its support in the set of possible parameterizations (the number of parameterizations for which the model supports a given decision). This was also put in practice through the stochastic multiattribute acceptability analysis [Angilella et al., 2015, Saint-Hilary et al., 2017], which samples a set of parameterizations and aggregates the results in order to obtain a decision support.

For instance, in a sorting problem between alternatives $a_1, ..., a_n$, the information that can be obtained in this way would be the proportion of models which rank alternative a_i in the first position, but also the proportion of models which rank a_i higher that a_j. This, under the assumption that the model is actually able to represent the preference information well, allows the DM to know to which extent the proposed decision is reliable.

U	H	CI	References
		✓	[Grabisch, 1995],[Grabisch et al., 2008] [Alavi et al., 2009],[Fallah Tehrani et al., 2012] [Hüllermeier and Fallah Tehrani, 2012],[Benabbou et al., 2017] [Havens and Anderson, 2018],[Bourdache et al., 2019]
✓			[Bous and Pirlot, 2013]
✓		✓	[Fallah Tehrani et al., 2014]
✓	✓	✓	[Huang et al., 2008],[Senge and Hüllermeier, 2011] (restricted to binary trees/2-dim CIs)
✓	✓	✓	Our Contribution

Table 3.1: Table of methods for learning diverse aspects of the UHCI models

3.9 Our Contribution

We have introduced in this part the basic notions that will serve as the basis for this thesis. In essence, we aim at leveraging the power of supervised machine learning and neural networks for learning the parameters of UHCI models, a work which was new before this thesis. Indeed, while the Choquet integral was learned before, even with marginal utilities, neither HCIs nor UHCIs were learned, limiting severely the representativity of the model. A review of references evoked in this chapter, about learning models that include Choquet integrals (CI), marginal utilities (U) and hierarchies (H) is presented in Table 3.9. Each time, we ticked the properties that are covered by the reference.

As we see, the state of the art in (H)CI learning is thus restricted to flat models, containing a single aggregation step. Furthermore these approaches, e.g. [Fallah Tehrani et al., 2012, Fallah Tehrani et al., 2014], hardly extend to HCI as they rely on a black-box sequential quadratic optimizer, and would face a non-convex optimization problem in the hierarchical case.

Our ambition is to address these limitations by learning the parameterization of 2-additive HCI models together with marginal utilities in an efficient way, assuming that the hierarchical structure be given. As will be shown, the framework that we developed, called NEUR-HCI, satisfies *by design* the formal (e.g. monotonicity) or semantic (e.g. based on the expert hierarchy) constraints, thereby enforcing the soundness, interpretabiliy and semantic meaningfulness of the eventual model. To the best of our knowledge, there is no other approach for learning HCI models with a general hierarchy.

On the other hand, the proposed methods for learning hierarchical models learn arguably different models, for different contexts. They indeed learn aggregators which are more general than Choquet Integral, as they also encompass

non-compensatory aggregation functions (t-norms or co-norms for instance), but are restricted to a binary setting (each aggregator has 2 inputs). This allows them to learn the hierarchy in a tractable way, which contrasts with our approach where the aim is to incorporate an expert-given hierarchy (based on domain knowledge). Moreover, each criterion can, in this approach, appear at several leaves of the tree, each with its own marginal utility. As a consequence, there might be a loss of interpretability, especially if the model is not bounded in size.

The main challenge is in learning such highly constrained models, which includes domain knowledge which must be strictly and formally enforced (rather than simply "approached", through classic regularizations). This allows for a fully-trained, understandable and predictable model, which a domain expert can easily verify and validate, or discard.

Our contribution is presented in two parts. First, Part III presents a proof of identifiability of the UHCI model; this proof of uniqueness lays the theoretical basis validating the attempts at learning such model, along with hinting at empirical satability of the learned model. Then Part IV presents the practical implementation of NEUR-HCI, our framework for learning a UHCIs model's aggregators and marginal utilities, while preserving all of its constraints.

Part III

Theoretical contributions

CHAPTER 4

INDENTIFIABILITY OF A UHCI WITH A FIXED HIERARCHY

Contents

4.1 General Considerations

4.1.1 Identifiability

The main contribution of this part is establishing the identifiability of UHCI models. The *identifiability* [Rothenberg, 1971] of a model is the uniqueness of its parameterization. That is, let $\mathcal{C} = \{\mathcal{F}_\theta, \theta \in \Theta\}$ be a family of functions defined on X, and with parameters in the parameter space Θ. Let F_θ (resp. $F_{\theta'}$) $\in \mathcal{C}$ be parameterized by θ (resp. θ'). Then $\mathcal{C}$ is identifiable if and only if: $\forall \mathbf{x} \in X, F_\theta(\mathbf{x}) = F_{\theta'}(\mathbf{x}) \Rightarrow \theta = \theta'$.

This subsection focuses on the case where the hierarchy of the UHCI is fixed, and establishes the identifiability property under this assumption. Let us consider two UHCIs that are equal everywhere on the input space, and which are assumed to have the same hierarchy denoted by $\mathcal{T}$. Proving Identifiability of a UHCI model with fixed hierarchy means that:

(A) the marginal utilities are equal on both models for each criterion;

(B) the fuzzy measures of each of the aggregators are equal between both models.

Under some assumptions, Subsections 4.1.3 and 4.2 establishes (A) and (B).

4.1.2 Motivation

In many application domains related to decision making and artificial intelligence at large, an essential requirement is to gain the users' trust [O'Neill, 2016]. To this end, the model must be *interpretable*, that is, the DMs must understand which criteria influence the decision, how, and to which extent; in other words, they must be able to trace back the assessment of an alternative to the criteria involved. In some cases, syntactic constraints (e.g. monotonicity) might be enforced to facilitate the interpretation of the model; in other cases, specific domain knowledge is available (e.g. implying some preferences w.r.t. some criteria, everything else being equal), and the model must comply with this prior knowledge. Naturally, the trust-worthiness of the model is all the more important in safety-critical contexts.

In order for the model's explanation to be trustable by a human DM, it is required that several conflicting interpretations could not be drawn from it, which would make the model confusing. One can now see why identifiability of the model is of the essence. Typically, ambiguities arise if the same utility function can be represented in different ways, preventing the DM from understanding the impact of each attribute; for instance, one model may suggest that a certain criterion is rather unimportant, while the other one concedes it a high relevance. In this case, which interpretation can the DM trust ? In such a case, the model becomes untrustable -and thus useless-.

Note that the identifiability property is also relevant in the machine learning context: the existence of a single solution brings several benefits to the learning [Paulino and Pereira, 1994, Ran and Hu, 2017].

The contribution of this part is to establish the identifiability of UHCIs, formally showing that, if two UHCIs are equal for all possible alternatives, then they have to have the same parameters, and thus the same interpretation. This result encompasses the identifiability of the marginal utilities from the raw criteria data, of the aggregation hierarchy, and of the aggregation parameters. Moreover,

it bridges a gap between both fields of Machine Learning − where powerful black-box models are learned with no general identifiability guarantees − and MCDA, where models are constrained and need to be interpretable. The UHCI model space thus offers an interesting trade-off for safety-critical applications, enabling the data-driven learning of interpretable and identifiable models.

In this chapter, we establish the UHCI identifiability when the hierarchy is fixed. Then, Chapter 5 establishes the identifiability of both the hierarchy and the parameters of the sought UHCI. Note that these results can be found in our paper [Bresson et al., 2021]. In both of these chapters, for readability reasons, we split the chapters into two main parts:

- the general idea of the proof, presenting assumptions, lemmas, algorithms and other necessary tools
- the actual proofs of said lemmas and theorems

4.1.3 Assumptions

To represent information on the local preferences restricted to single features (attributes), it is usual to impose some assumptions on the marginal utilities. Note that the marginal utilities used in practice are either monotonic or bitonic (either single-peaked or single-valleyed). In Example 1, it is increasing w.r.t. criterion 1 (surface area) and single-peaked w.r.t. criterion 4 (distance to large road).

The continuity of u_i is desirable to avoid a non-stable behavior of the local and hence the global utility. To have meaningful relative importance degrees among criteria, one also imposes the marginal utilities to be normalized. Moreover, a very common assumption is that the smallest possible utility is 0 (a value suggesting that the corresponding criterion is not met at all), and the largest one is 1 (the criterion is satisfied). For convenience, we impose that these extreme values be reached (possibly asymptotically) for a given value on attribute i. Overall, the assumptions are summarized as follows:

$$\forall i \in N \quad u_i \text{ is continuous on } \overline{\mathbb{R}} \tag{4.1}$$

$$\forall i \in N \quad \inf_{x_i \in \overline{\mathbb{R}}} u_i(x_i) = 0 \tag{4.2}$$

$$\forall i \in N \quad \sup_{x_i \in \overline{\mathbb{R}}} u_i(x_i) = 1 \tag{4.3}$$

All marginal utilities evoked in the remainder of this chapter are assumed to respect constraints (4.1), (4.2) and (4.3).

Assume in Figure 2.2 that node 9 has no effect in the aggregation 11. Then one can modify the parameters of aggregations 8 and 9, with no consequence on

the global score. This illustrates that identifiability cannot be obtained if there are useless criteria. We formalize this in the following way. We say that no criterion in N is *useless* if

$$\forall i \in N \; \exists x_i, y_i \in X_i, \boldsymbol{z} \in X, \\ \mathcal{F}(x_i, \boldsymbol{z}_{-i}) \neq \mathcal{F}(y_i, \boldsymbol{z}_{-i}) \tag{4.4}$$

where $(x_i, \boldsymbol{z}_{-i})$ is the alternative whose value is x_i on attribute i, and z_j on all other attributes $j \neq i$. We introduce the following definition on FMs.

Definition 8. *Consider a FM μ on N. A criterion $i \in N$ is said to be* degenerate *if $\mu(S \cup \{i\}) = \mu(S)$ for every $S \subseteq N \setminus \{i\}$. A fuzzy-measure is said to be* non-degenerate *if there is no degenerate criterion.*

A first result (inspired from [Labreuche, 2018]) is:

Lemma 1. *Relation (4.4) holds for $\mathcal{F}$ with marginal utility functions if and only all aggregators $\mathcal{A}_k$ for $k \in V$ have a non-degenerate measure.*

4.2 Showing Identifiability with a Fixed Hierarchy

The uniqueness of the representation of the UHCI model on a fixed hierarchy follows from Lemma 1:

Theorem 1. *Let $\mathcal{F}$ and $\mathcal{F}'$ be two UHCIs with same hierarchy $\mathcal{T} = \langle r, M, \mathrm{Ch} \rangle$, and assume that they involve different fuzzy measures and utilities. Assuming that (4.4) holds for $\mathcal{F}$ and $\mathcal{F}'$, and that the following relation is satisfied*

$$\forall \mathbf{x} \in X, \qquad \mathcal{F}(\mathbf{x}) = \mathcal{F}'(\mathbf{x}), \tag{4.5}$$

then $\mathcal{F}$ and $\mathcal{F}'$ have the same parameterization, that is:

- $\forall i \in N$, $\forall x_i \in X_i$, $u_i(x_i) = u'_i(x_i)$
- $\forall k \in V$, $\mu_k = \mu'_k$

The proof is organized as follows (see the detailed proof in Section 4.4): First, $\mathcal{F}$ and $\mathcal{F}'$ can be written:

$$\mathcal{F} : \mathbf{x} \mapsto C_{\mu_r}(\mathcal{F}_{r_1}(\mathbf{x}_{r_1}), \ldots, \mathcal{F}_{r_{d(r)}}(\mathbf{x}_{r_{d(r)}})), \\ \mathcal{F}' : \mathbf{x} \mapsto C_{\mu'_r}(\mathcal{F}'_{r_1}(\mathbf{x}_{r_1}), \ldots, \mathcal{F}'_{r_{d(r)}}(\mathbf{x}_{r_{d(r)}})),$$

with $\mathrm{Ch}(r) = \{r_1, ..., r_{d(r)}\}$. Using this form, it is enough to show these two properties:

(*) $\mu_r = \mu'_r$,

(**) $\forall k \in \mathrm{Ch}(r), \forall \mathbf{x} \in X_k, \quad \mathcal{F}_k(\mathbf{x}) = \mathcal{F}'_k(\mathbf{x})$.

Indeed, if we prove this for the root, then the proof applies on the children as well, as they are all UHCI. We can then propagate the result from the root to its children, and so on. When we reach a leaf i, we show that we obtain $u_i = u'_i$. We distinguish three cases, represented each by a lemma below.

Lemma 2. *If* $\forall k \in \mathrm{Ch}(r)$, $\forall \mathbf{x} \in X_k$, $\mathcal{F}_k(\mathbf{x}) = \mathcal{F}'_k(\mathbf{x})$, *then we have* $\mu_r = \mu'_r$.

Lemma 2 means that (**) $\Rightarrow$ (*).

Let $\overline{B}_k = \{\mathbf{x}_k \in X_k \ : \ \mathcal{F}_k(\mathbf{x}) = 1\}$ and $\underline{B}_k = \{\mathbf{x}_k \in X_k \ : \ \mathcal{F}_k(\mathbf{x}) = 0\}$. The next two steps depend on the intersections of $\underline{B}_k \cap \underline{B'}_k$ and $\overline{B}_k \cap \overline{B'}_k$ of all children $k \in \mathrm{Ch}(r)$.

Lemma 3. *If,* $\forall k \in \mathrm{Ch}(r)$,
$\left[(\underline{B}_k \cap \underline{B'}_k \neq \emptyset)\right.$ *and* $\left.(\overline{B}_k \cap \overline{B'}_k \neq \emptyset)\right]$, *then we have (**).*

Lemma 3, shows that if all $\mathcal{F}_k$ and $\mathcal{F}'_k$ have non-disjoint support for 0 and for 1, then we have (**); by Lemma 2, we also have (*). We thus have identifiability.

Lemma 4. *Assume there exists* k *in* $\mathrm{Ch}(r)$ *such that* $\underline{B}_k \cap \underline{B'}_k = \emptyset$ *or* $\overline{B}_k \cap \overline{B'}_k = \emptyset$. *Then,* $\mathcal{F} \neq \mathcal{F}'$.

Lemma 4 shows that, should the assumptions of Lemma 3 be violated, then $\mathcal{F}$ is necessarily different from $\mathcal{F}'$. As a consequence, we have an equivalence between the assumptions of Lemma 3 and that of Theorem 1; we have thus shown that, given two equal UHCIs with the same hierarchy, they have necessarily the same marginal utilities, and the same aggregations. Then, we can conclude:

Proof of Theorem 1:

$$\begin{aligned} \mathcal{F} = \mathcal{F}' &\Rightarrow \forall k \in \mathrm{Ch}(r), \\ \left[(\underline{B}_k \cap \underline{B'}_k \neq \emptyset) \text{ and } (\overline{B}_k \cap \overline{B'}_k \neq \emptyset)\right] &\text{ by Lemma 4} \\ \Rightarrow \forall k \in \mathrm{Ch}(r), \mathcal{F}_k = \mathcal{F}'_k &\text{ by Lemma 3} \\ \Rightarrow \mu_r = \mu'_r &\text{ by Lemma 2} \end{aligned}$$

The proof is completed. □

4.3 Conclusion

We have, in this chapter, shown that, given a UHCI with a given hierarchy, there is unicity of both its fuzzy measures and marginal utilities, under mild assumptions. The proofs of all lemmas and theorems evoked are given thereafter in Section 4.4. This is a first interesting result, as it allows to ensure the unicity of the interpretation of a given HCI model. There now remains to show that this unicity still holds when the hierarchy is left unfixed, which is done in the next chapter.

4.4 Proofs

Proof of Lemma 2: Let $S \subseteq M$ a set of nodes, we write $\mathrm{Lf}(S) = \bigcup_{k \in S} \mathrm{Lf}(k)$. We use the following compound notation. Let $\mathbf{x}, \mathbf{x}'$ be two vectors in X, and let $S, S' \subseteq M$ so that $(\mathrm{Lf}(S), \mathrm{Lf}(S'))$ is a partition of N. Then we write $\mathbf{x} = (\mathbf{x}_S, \mathbf{x}'_{S'})$ the vector s.t. $\mathbf{x}_i = \begin{cases} \mathbf{x}_i & \text{if } i \in \mathrm{Lf}(S) \\ \mathbf{x}'_i & \text{if } i \in \mathrm{Lf}(S') \end{cases}$

Let $\mathbf{Z} \in \prod_{k \in \mathrm{Ch}(r)} \underline{B}_k$, and $\mathbf{O} \in \prod_{k \in \mathrm{Ch}(r)} \overline{B}_k$.

Let S be an arbitrary subset of $\mathrm{Ch}(r)$. By definition, we have $\mathcal{F}(\mathbf{O}_S, \mathbf{Z}_{\mathrm{Ch}(r)\setminus S}) = C_{\mu_r}(1_S, 0_{\mathrm{Ch}(r)\setminus S})$. Thus $\mathcal{F}(\mathbf{O}_S, \mathbf{Z}_{\mathrm{Ch}(r)\setminus S}) = \mu_r(S)$. In the same way, we obtain $\mathcal{F}'(\mathbf{O}_S, \mathbf{Z}_{\mathrm{Ch}(r)\setminus S}) = \mu'_r(S)$.

As the values of $\mathcal{F}$ and $\mathcal{F}'$ are equal for any $\mathbf{x} \in X$, we have $\mu_r(S) = \mu'_r(S)$. This applies for all $S \subseteq \mathrm{Ch}(r)$, thus Property (*) holds. □

Proof of Lemma 3: Let $k \in \mathrm{Ch}(r)$. By the assumption of the lemma, we can construct $\mathbf{Z} \in \prod_{k \in \mathrm{Ch}(r)} \underline{B}_k \cap \underline{B'}_k$, and $\mathbf{O} \in \prod_{k \in \mathrm{Ch}(r)} \overline{B}_k \cap \overline{B'}_k$. Since μ_r is by hypothesis non-degenerate, we have that $\exists S_k \subseteq \mathrm{Ch}(r) \setminus \{k\}$ s.t. $\mu_r(S_k \cup \{k\}) - \mu_r(S_k) > 0$. Thus, we have: $\forall \mathbf{x}_k \in X_k$

$$\begin{aligned} &\mathcal{F}(\mathbf{O}_{S_k}, \mathbf{Z}_{\mathrm{Ch}(r)\setminus(S_k \cup \{k\})}, \mathbf{x}_k) \\ &= C_{\mu_r}(1_{S_k}, 0_{\mathrm{Ch}(r)\setminus(S_k \cup \{k\})}, \mathcal{F}_k(\mathbf{x}_k)) \\ &= \mathcal{F}_k(\mathbf{x}_k)(\mu_r(S_k \cup \{k\}) - \mu_r(S_k)) + \mu_r(S_k) \end{aligned}$$

Likewise for $\mathcal{F}'$, thus, we have the following equality:

$$\begin{aligned} &\mathcal{F}_k(\mathbf{x}_k)(\mu_r(S_k \cup \{k\}) - \mu_r(S_k)) + \mu_r(S_k) \\ &= \mathcal{F}'_k(\mathbf{x}_k)(\mu'_r(S_k \cup \{k\}) - \mu'_r(S_k)) + \mu'_r(S_k) \end{aligned}$$

Since $\mu_r(S_k \cup \{k\}) - \mu_r(S_k) > 0$ by definition of S_k, we obtain:

$$\exists \alpha \in \mathbb{R}_+, \beta \in \mathbb{R}, \forall \mathbf{x}_k \in X_k, \mathcal{F}_k(\mathbf{x}_k) = \alpha \mathcal{F}'_k(\mathbf{x}_k) + \beta$$

Applying this equality in $\mathbf{Z}_k$ and $\mathbf{O}_k$, we get $\alpha = 1$ and $\beta = 0$.

$$\mathcal{F}_k(\mathbf{Z}_j) = \alpha\mathcal{F}'_k(\mathbf{Z}_j) + \beta \Rightarrow 0 = 0 + \beta \Rightarrow \beta = 0$$

$$\mathcal{F}_k(\mathbf{O}_j) = \alpha\mathcal{F}'_k(\mathbf{O}_k) + \beta \Rightarrow 1 = 1\alpha \Rightarrow \alpha = 1$$

We have thus shown that the assumption on this case implies that $\forall k \in \mathrm{Ch}(r), \mathcal{F}_k = \mathcal{F}'_k$. □

We define $\underline{u}_i$ and $\overline{u}_i$ to the marginal utility. For $i \in N$, we write $\underline{u}_i = \{x_i \in X_i : u_i(x_i) = 0\}$ and $\overline{u}_i = \{x_i \in X_i : u_i(x_i) = 1\}$. Note that, in some cases, these values can be reached asymptotically. In these cases, $\underline{u}_i$ (resp $\overline{u}_i$) might only contain $-\infty$ or $+\infty$.

Property 1. *Let $k \in \mathrm{Ch}(r)$, such that $\overline{B}_k \cap \overline{B'}_k = \emptyset$ (resp. $\underline{B}_k \cap \underline{B'}_k = \emptyset$). Then there exists a leaf $i \in \mathrm{Lf}(k)$ such that $\overline{u}_i \cap \overline{u}'_i = \emptyset$ (resp. $\underline{u}_i \cap \underline{u}'_i = \emptyset$).*

Proof : We show the first result by contradiction. Let $k \in \mathrm{Ch}(r)$, such that $\overline{B}_k \cap \overline{B'}_k = \emptyset$. We assume that $\forall i \in \mathrm{Lf}(k)$, $\exists\, \overline{x}_i \in \overline{u}_i \cap \overline{u}'_i$. Then there exists $\overline{\mathbf{x}}_k = (\overline{x}_1, \cdots, \overline{x}_{|\mathrm{Lf}(k)|})$. Thus $\mathcal{F}_k(\overline{\mathbf{x}}_k) = \mathcal{A}_k(1, \ldots, 1) = 1$. Using the same argument, we can obtain : $\mathcal{F}'_k(\overline{\mathbf{x}}_k) = 1$. Thus, $\overline{\mathbf{x}}_k \in \overline{B}_k \cap \overline{B'}_k$, hence the contradiction.

The very same reasoning can be applied for the case $\underline{B}_k \cap \underline{B'}_k = \emptyset$. □

Proof of Lemma 4: We prove this lemma by contradiction, by assuming $\mathcal{F} = \mathcal{F}'$. Then, let $i \in \mathrm{Lf}(k)$ such that $\overline{u}_i \cap \overline{u}'_i = \emptyset$ or $\underline{u}_i \cap \underline{u}'_i = \emptyset$, which exists by Property 1. There is thus an interval $I = [\alpha_i, \beta_i] \subseteq X_i$ such that u_i and u'_i have opposite monotonicity on I. This can easily be verified by enumerating the cases. WLOG, we assume that u_i is decreasing on I while u'_i is non-decreasing.

Let $S \subseteq N \setminus \{i\}$. Let $\mathbf{x}^S \in \prod_{j \in S} \overline{u}_j \times \prod_{i \in N \setminus S} \underline{u}_j$. Then, by monotonicity of HCIs, $u_i(\alpha_i) < u_i(\beta_i)$ implies:

$$\begin{aligned} &\mathcal{F}_k(\alpha_i, \mathbf{x}^S_{\mathrm{Lf}(k)\setminus\{i\}}) \leq \mathcal{F}_k(\beta_i, \mathbf{x}^S_{\mathrm{Lf}(k)\setminus\{i\}}) \\ &\Rightarrow \mathcal{F}(\alpha_i, \mathbf{x}^S_{N\setminus\{i\}}) \leq \mathcal{F}(\beta_i, \mathbf{x}^S_{N\setminus\{i\}}) \end{aligned}$$

Likewise, we have: $\mathcal{F}'(\alpha_i, \mathbf{x}^S_{N\setminus\{i\}}) \geq \mathcal{F}'(\beta_i, \mathbf{x}^S_{N\setminus\{i\}})$ as u'_i is non-decreasing on $[\alpha_i, \beta_i]$. Since $\mathcal{F}$ and $\mathcal{F}'$ are equal everywhere, we thus have:

$$\begin{aligned} &\forall S \subseteq N \setminus \{i\}, \\ &\quad \mathcal{F}\left(\alpha_i, \mathbf{x}^S_{N\setminus\{i\}}\right) = \mathcal{F}\left(\beta_i, \mathbf{x}^S_{N\setminus\{i\}}\right) \end{aligned}$$

We show that this equation leads to a contradiction, by building an $S \subseteq N$ such that we do not have the equality.

We denote by $\pi = \{\pi_1, \ldots, \pi_t\}$ the unique path from root r to leaf i, with t the depth of i in the tree. We order them so that $\pi_1 = r$ and $\forall j \in \{2, \ldots, t\}, \pi_j \in \mathrm{Ch}(\pi_{j-1})$. This means that $\pi_t = i$.

For each j in $\{1, \ldots, t-1\}$, let S_j be a subset of nodes in $\mathrm{Ch}(\pi_j) \setminus \{\pi_{j+1}\}$ such that $\mu_{\pi_j}(S_j) > \mu_{\pi_j}(S_j \cup \{\pi_{j+1}\})$. The non-degenerateness property implies that such sets exist. We denote by $S^k = \bigcup_{j=1}^{t-1} S_j$.

Now, let $\mathbf{v}^\alpha$ be a vector in X such that : $\begin{cases} \mathbf{v}^\alpha = \alpha_i \\ \mathbf{v}_i^\alpha \in \overline{u}_i & \text{if } \exists g \in S^k, i \in \mathrm{Lf}(g) \\ \mathbf{v}_i^\alpha \in \underline{u}_i & \text{otherwise} \end{cases}$

and $\mathbf{v}_\beta = (\mathbf{v}^\alpha_{N \setminus k}, \beta_i)$.

We write $d_j = \mathrm{Ch}(\pi_j) \setminus (S_j \cup \{\pi_{j+1}\})$. Then, at node π_j, we have:

$$\begin{aligned} \mathcal{F}_{\pi_j}(\mathbf{v}^\alpha_{\pi_j}) &= C_{\mu_{\pi_j}}(\mathcal{F}_{\pi_{j+1}}(\mathbf{v}^\alpha_{\pi_{j+1}}), 1_{S_j}, 0_{d_j}) \\ &= \mathcal{F}_{\pi_{j+1}}\left(\mathbf{v}^\alpha_{\pi_{j+1}}\right)\gamma_j + \delta_j \end{aligned}$$

with $\gamma_j = \mu_{\pi_j}(S_j \cup \{\pi_{j+1}\}) - \mu_{\pi_j}(S_j)$ and $\delta_j = \mu_{\pi_j}(S_j)$.

when $j = t-1$, we have:

$$\mathcal{F}_{\pi_{t-1}}(\mathbf{v}^\alpha_{\pi_{t-1}}) = \gamma_{t-1} u_i(\alpha_i) + \delta_{t-1}$$

$$\begin{aligned} \mathcal{F}_{\pi_{t-2}}(\mathbf{v}^\alpha_{\pi_{t-2}}) &= \gamma_{t-2}\mathcal{F}_{\pi_{t-1}}\left(\mathbf{v}^\alpha_{\pi_{t-1}}\right) + \delta_{t-2} \\ &= \gamma_{t-2}(\gamma_{t-1} u_i(\alpha_i) + \delta_{t-1}) + \delta_{t-2} \end{aligned}$$

And so on. Since $\forall j \in \{1, ..., t\}, \gamma_j > 0$ by construction of S_j, and $\delta_j \geq 0$, we have $\mathcal{F}_{\pi_1}(\mathbf{v}^\alpha_{\pi_t}) = \Gamma u_i(\alpha_i) + \Delta$ with $\Gamma > 0$ and $\Delta \geq 0$ by composition of affine functions with a strictly increasing coefficient. Likewise, for $\mathbf{v}^\beta$, we obtain $\mathcal{F}_{\pi_1}(\mathbf{v}^\beta_{\pi_t}) = \Gamma u_i(\beta_i) + \Delta$. Nonetheless:

$$\begin{aligned} u_i(\alpha_i) < u_i(\beta_i) &\Rightarrow \Gamma u_i(\alpha_i) + \Delta < \Gamma u_i(\beta_i) + \Delta \\ &\Rightarrow \mathcal{F}(\mathbf{v}^\alpha_{\pi_t}) < \mathcal{F}_r(\mathbf{v}^\beta_{\pi_t}) \end{aligned}$$

We have built a set $S \subseteq N$ such that Eq. (4.5) does not hold. Thus $\mathcal{F} \neq \mathcal{F}'$ and this yields to the contradiction. □

CHAPTER 5

GENERAL IDENTIFIABILITY OF A UHCI

Contents

5.1 Showing Identifiability with a Free Hierarchy

The aim of this chapter is to show the identifiability of the UHCI model in the general case, that is, when the hierarchy, the aggregation functions and the marginal utility functions are unknown. Let us start with an example to give the intuition of the approach.

Example 26. *Consider three criteria organized as in Figure 5.1. We assume the models* $a_4 = \frac{a_1+\min(a_1,a_2)}{2}$ *and* $a_5 = \frac{a_3+\min(a_3,a_4)}{2}$ *and identity marginal utility*

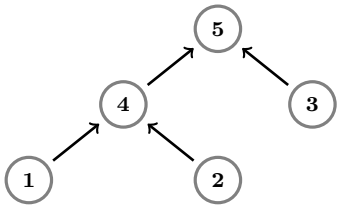


Figure 5.1: Example of a tree.

functions. As we will see in this section, a_5 *is piecewise affine with respect to* a_1, a_2, a_3 *and is composed of the following affine parts* a_3, $\frac{a_1}{2} + \frac{a_3}{2}$ *and* $\frac{a_3}{2} + \frac{a_1+a_2}{4}$ *which are separated by the frontiers*

$$a_1 = a_2 \ , \ a_1 = a_3 \quad \textit{and} \quad \frac{a_1 + a_2}{2} = a_3. \tag{5.1}$$

We note that, at the root of the hierarchy presented in 5.1, the root (node 5*), has two children, each of which define a subtree:*

- *the subtree whose root is* 4*, and with leaves* 1 *and* 2
- *the subtree composed of the single node* 3*, which is a leaf, and thus a native criterion*

We can thus partition the leaves depending on the subtree to which they belong, yielding the subsets $\{1, 2\}$ *and* $\{3\}$*. This partition can be found again in the equation for the third separation border in Equation* (5.1)*. Indeed, the left-hand side of the equation is a convex sum involving criteria* 1 *and* 2*, while the right-hand side is a convex sum involving only criterion* 3*.*

As illustrated in this example, the tree structure can be recovered from the expression of the separation frontiers. Theorem 1 can then be applied, and yields the uniqueness of both the aggregator weights and the marginal utility functions. So the major ingredient to show identifiability of the UHCI model is to have a good characterization of the separation frontiers (Section 5.1.1). In Section 5.1.2, we present an algorithm Alg_1 which, given a UHCI under the form of its marginal utilities, hierarchy and set of fuzzy measures on its non-leaf nodes, yields the set of separation frontiers. In Section 5.1.3, we present an algorithm Alg_2 which, given a set of separation frontiers of a UHCI model, constructs a tree compatible with these frontiers. Finally, in Section 5.1.4, we exhibit two conditions on the hierarchy $\mathcal{T}$ and aggregation of a UHCI $\mathcal{F}$. Under those conditions, $\text{Alg}_2(\text{Alg}_1(\mathcal{F})) = \mathcal{T}$; that is, $\mathcal{T}$ is the only hierarchy that can parameterize $\mathcal{F}$.

Towards a generalization of the Choquet integral

By Eq. (2.10), a CI is a piecewise affine function, involving subdomains where the CI returns a weighted sum, and the domains being separated by hyperplanes of the form $x_i \geq x_j$. While these domains make sense in the view of the idempotency property, one would like to generalize them.

It is shown in [Grabisch et al., 2009], based on [Ovchinnikov, 2002], that any non-decreasing, and positively homogeneous piecewise linear function can be represented by a network of interconnected CIs. The statement in question is that the three following properties are equivalent:

- $\mathcal{F}$ is a multilevel Choquet integral;
- $\mathcal{F}$ is a 2-level Choquet integral, with all capacities of the first level being additive, and the capacity of the second level being 0–1 valued;
- $\mathcal{F}$ is nondecreasing and positively homogeneous piecewise linear.

While this result shows the generality of interconnected CIs, the lack of structure on these CIs might harm the overall interpretability, all the more so as there is no restriction on number of CIs, possibly yielding very large models. Moreover, as each CI aggregates all inputs, the traceability of the criteria impact and their interactions is lost. The model needs to be inspected in every detail to be understood. Moreover, this results shows that such DAG-CIs are not identifiable, as they can be re-written as a 2-level CI.

As a consequence, we focus on tree-like structures, relaxing the 2-level condition, but effectively constraining the model to sparse and interpretable models, with a width naturally limited by the number of criteria.

5.1.1 Structure of the UHCI

From the definition of the CI (given in Equation (2.10), in Chapter 2), we see that the CI (and thus the HCI) is a continuous, piecewise-linear function w.r.t. its inputs. It is, more precisely, a piecewise-convex combination, as the weights in all regions sum to one and are non-negative.

We consider here a UHCI model $\mathcal{F}$ characterized by a tree $\mathcal{T} = \langle r, M, \mathrm{Ch} \rangle$, a set of FM $\{\mu_k\}_{k \in V}$ for its aggregations, and a set of marginal utility functions $\mathcal{U} = \{u_1, \cdots, u_n\}$. For an HCI model, identifiability is obtained by analyzing the linear separation borders between linear parts. For a UHCI, we impose the following regularity constraint on the marginal utilities:

$$\forall i \in N, \quad u_i \text{ is piecewise } C^1. \tag{5.2}$$

This is a reasonable assumption, as it encompasses most used models, such as sigmoids or piecewise affine functions. The UHCI is thus now a piecewise-C^1 function, and our strategy is to analyze the separation frontiers between the C^1 parts.

We also maintain assumptions (4.1), (4.2), (4.3) on the marginal utilities, as in Section 4.1.1.

$\mathcal{F}$ can thus be described by a finite number of C^1 functions $\mathcal{G} = \{g_1, \ldots, g_m\}$. It is, in particular, a piecewise convex combination of the output of the marginal utilities:

$$\forall g \in \mathcal{G}, \quad g(\mathbf{x}) = \sum_{i=0}^{n} w_i^g u_i(x_i),$$

with non-negative weights that sum to one.

Space X can thus be split into a finite number of regions $R_1, ..., R_m$, such that $\forall \mathbf{x} \in R_i,\ \mathcal{F}(\mathbf{x}) = g_i(\mathbf{x})$. $\mathcal{F}$ is thus C^1 on each of these regions, and we call $\mathcal{H}$ the set of frontiers between each of these regions. $\mathcal{F}$ is also continuous on X.

For a node $k \in M$, we call $\mathcal{G}_k$ the set of C^1 functions that $\mathcal{F}_k$ can take, where these C^1 functions depend only on $\mathbf{x}_k$, and $\mathcal{H}_k$ the set of separation frontiers between the C^1 regions of $\mathcal{F}_k$ that are induced by node k.

Note that $\mathcal{G} = \mathcal{G}_r$, but $\mathcal{H} = \bigcup_{k \in M} \mathcal{H}_k$.

Lemma 5. *For a leaf node $i \in N$, the separation frontiers of $\mathcal{H}_i$ are of the form $x_i = \theta$ with $\theta \in X_i$.*

For an aggregation node $k \in V$, the separation frontiers of $\mathcal{H}_k$ take the form of linear equations of the marginal utility functions of the attributes: $\sum_{l \in K} w_l u_l(x_l) = 0$ with $K \subseteq \mathrm{Lf}(k)$ and $w_l \neq 0$ for all $l \in K$. Then there exist $k', k'' \in \mathrm{Ch}(k)$ with $k' \neq k''$ such that we have $\{l \in K \ :\ w_l > 0\} \subseteq \mathrm{Lf}(k')$ and $\{l \in K \ :\ w_l < 0\} \subseteq \mathrm{Lf}(k'')$.

$\mathcal{H}$ thus contains separation frontiers of two forms. We call form 1 the frontiers written as $x_i = \theta$, i.e. induced by a leaf. We call form 2 the frontiers of the type $\sum_{l \in K} w_l\, u_l(x_l) = 0$, i.e. induced by an aggregation node.

5.1.2 Construction of the Set of Separation Frontiers from the UHCI Model

We have just seen that a HCI model is a piecewise linear function. As the marginal utility functions are piecewise C^1, we conclude that a UHCI model is a piecewise C^1 function.

We present Algorithm 1 in this section. This algorithm takes as an input a UHCI, characterized by its hierarchy $\mathcal{T} = \langle r, M, \mathrm{Ch} \rangle$, its set of the fuzzy measures

of all of its aggregators and its marginal utility functions. It then computes $\mathcal{H}$. Note that this algorithm also computes $\mathcal{G}$ as an internal variable. The approach is constructive, building the two sets $\mathcal{G}_k$ and $\mathcal{H}_k$ for every $k \in M$ in a bottom up manner from the leaves to the root.

Let $k \in V$. For FM μ_k, we write $\mathcal{S}_k$ the set of subsets having a non-zero Möbius coefficient in Eq. (2.11). As stated a_k is a piecewise linear function of the $u_i(\mathbf{x}_i)$ for $i \in \mathrm{Lf}(k)$.

We now illustrate Alg_1 in an example.

Example 27. *Consider the tree given by Figure 5.2 and a UHCI model defined by* $a_6 = \frac{1}{2}\min(a_3, a_5) + \frac{1}{2}\min(a_4, a_5)$, $a_5 = \frac{1}{2}a_1 + \frac{1}{2}\min(a_1, a_2)$, $u_k(x_k) = {x_k}^{k+1}$ *for* $k = 1, 2, 3$, *and* $u_4(x_4) = \sqrt{x_4}$ *if* $x_4 \leq \frac{1}{4}$ *and* $u_4(x_4) = \frac{2x_4+1}{3}$ *otherwise, with* $X_1 = \cdots = X_4 = [0, 1]$. *Utility function* u_4 *is continuous and has two* C^1 *segments.*

The execution of Algorithm 1 gives:

- *At* $k = 1, 2, 3$: $\mathcal{G}_k = \{{x_k}^{k+1}\}$ *and* $\mathcal{H}_k = \emptyset$.
- *At* $k = 4$: $\mathcal{G}_4 = \{\sqrt{x_4}; \frac{2x_4+1}{3}\}$ *and* $\mathcal{H}_4 = \{x_4 = \frac{1}{4}\}$.
- *At* $k = 5$: $\mathcal{E}_5 = \{(1, 2)\}$ *so that the connected component is* $\{1, 2\}$. *Then* a_5 *can take the expressions* a_1 *and* $\frac{a_1+a_2}{2}$ *depending on whether* $a_1 \leq a_2$. *Hence* $\mathcal{G}_5 = \{{x_1}^2\,;\, \frac{{x_1}^2+{x_2}^3}{2}\}$ *and* $\mathcal{H}_5 = \{{x_1}^2 = {x_2}^3\}$.
- *At* $k = 6$: $\mathcal{E}_6 = \{(3, 5), (4, 5)\}$ *so that the connected component is* $\{3, 4, 5\}$. *Then* a_6 *can take the expressions* a_5, $\frac{a_3+a_5}{2}$, $\frac{a_5+a_4}{2}$ *and* $\frac{a_3+a_4}{2}$ *depending on whether* $a_5 \leq a_3$ *and* $a_5 \leq a_4$. *Hence* $\mathcal{G}_6$ *is composed of*
  - ${x_1}^2$ *[for* $a_1 < a_2, a_5 < a_3, a_5 < a_4$*]*
  - $\frac{{x_1}^2+{x_2}^3}{2}$ *[for* $a_1 > a_2, a_5 < a_3, a_5 < a_4$*]*
  - $\frac{{x_1}^2}{2} + \frac{{x_3}^4}{2}$ *[for* $a_1 < a_2, a_5 > a_3, a_5 < a_4$*]*
  - $\frac{{x_1}^2+{x_2}^3}{4} + \frac{{x_3}^4}{2}$ *[for* $a_1 > a_2, a_5 > a_3, a_5 < a_4$*]*
  - $\frac{{x_1}^2}{2} + \frac{\sqrt{x_4}}{2}$ *[for* $a_1 < a_2, a_5 < a_3, a_5 > a_4, a_4 < \frac{1}{4}$*]*
  - $\frac{{x_1}^2+{x_2}^3}{4} + \frac{\sqrt{x_4}}{2}$ *[for* $a_1 > a_2, a_5 < a_3, a_5 > a_4, a_4 < \frac{1}{4}$*]*
  - $\frac{{x_1}^2}{2} + \frac{2x_4+1}{6}$ *[for* $a_1 < a_2, a_5 < a_3, a_5 > a_4, a_4 > \frac{1}{4}$*]*
  - $\frac{{x_1}^2+{x_2}^3}{4} + \frac{2x_4+1}{6}$ *[for* $a_1 > a_2, a_5 < a_3, a_5 > a_4, a_4 > \frac{1}{4}$*]*
  - $\frac{{x_3}^4}{2} + \frac{\sqrt{x_4}}{2}$ *[for* $a_5 > a_3, a_5 > a_4, a_4 < \frac{1}{4}$*]*
  - $\frac{{x_3}^4}{2} + \frac{2x_4+1}{6}$ *[for* $a_5 > a_3, a_5 > a_4, a_4 > \frac{1}{4}$*]*

Function $\mathrm{Alg}_1(\mathcal{T}, \{\mu_k\}_{k\in M}, \{u_k\}_{k\in N})$

While some nodes remain untreated
  Let $k \in M$ s.t. k has no untreated children
  If k is a leaf
    $\mathcal{G}_k \coloneqq$ set of C^1 expressions of u_k
    $\mathcal{H}_k \coloneqq$ set of constraints $x_k = \theta$ where θ is a transition between two C^1 domains of u_k
  Else
    $\mathcal{E}_k \coloneqq \{(l, l') \,:\, \exists S \in \mathcal{S}_k \text{ with } \{l, l'\} \subseteq S\}$
    For all C connected components of the graph $\langle \mathrm{Ch}(k), \mathcal{E}_k \rangle$:
      Compute the linear model of a_k as a function of the $a_{\mathrm{Ch}(k)}$, by transforming terms $\min_{i\in A} a_i$ into a_i for an index $i \in A$ depending on the order among the variables. This yields $\sum_{l\in K} w_l\, a_l$ with $K \subseteq \mathrm{Ch}(k)$
      For all term a_l, with $l \in K$
        For all expression $\sum_{j\in \mathrm{Lf}(l)} w_j^l\, u_j(x_j)$ in $\mathcal{G}_l$
          Add $\sum_{l\in K} \sum_{j\in \mathrm{Lf}(l)} w_l\, w_j^l\, u_j(x_j)$ to $\mathcal{G}_k$
  For all $(l, l') \in \mathrm{Ch}(k)^2$ s.t. $\exists S \in \mathcal{S}_k : \{l, l'\} \subseteq S$
    For all $\sum_{j\in \mathrm{Lf}(l)} w_j^l\, u_j(x_j) \in \mathcal{G}_l$
      For all $\sum_{j\in \mathrm{Lf}(l')} w_j^{l'}\, u_j(x_j) \in \mathcal{G}_{l'}$
        $\mathcal{H}_k \leftarrow \mathcal{H}_k \cup \Big\{ \mathbf{x} \in X \,:\, \sum_{j\in \mathrm{Lf}(l)} w_j^l\, u_j(x_j) = \sum_{j\in \mathrm{Lf}(l')} w_j^{l'}\, u_j(x_j) \Big\}$
  Add $\mathcal{H}_k$ to $\mathcal{H}$
Return $\mathcal{H}$

Algorithm 1: Obtaining $\mathcal{H}$ (and also $\mathcal{G}$) from a UHCI

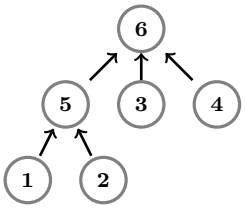


Figure 5.2: Example of a tree.

and $\mathcal{H}_6 = \{{x_1}^2 = {x_3}^4\,;\ \frac{{x_1}^2+{x_2}^3}{2} = {x_3}^4\,;\ {x_1}^2 = \sqrt{x_4}\,;\ {x_1}^2 = \frac{2\,x_4+1}{3}\,;\ \frac{{x_1}^2+{x_2}^3}{2} = \sqrt{x_4}\,;\ \frac{{x_1}^2+{x_2}^3}{2} = \frac{2\,x_4+1}{3}\}$.

Finally $\mathcal{H} = \mathcal{H}_4 \cup \mathcal{H}_5 \cup \mathcal{H}_6$. □

The next result shows that Algorithm 1 computes the set $\mathcal{G}$ of C^1 functions that $\mathcal{F}$ takes, and returns the set of separation frontiers $\mathcal{H}$ between these functions.

Lemma 6. *Algorithm 1 computes the correct values of the set of piecewise* C^1 *functions* $\mathcal{G}_k$ *and returns the correct set of separation frontiers* $\mathcal{H}_k$*, for each* $k \in M$*.*

Lemma 6 states, in particular, that the output of Alg_1 depends only on the values of the UHCI as a piecewise-C^1 function. The following corollary is a simple rewriting.

Corollary 1. *Let* $\mathcal{F}$ *(resp.* $\mathcal{F}'$*) be a UHCI defined on hierarchy* $\mathcal{T}$ *(resp.* $\mathcal{T}'$*), a set of FMs* ν *(resp.* ν'*) and a set of marginal utility functions* $\mathcal{U}$ *(resp.* $\mathcal{U}'$*). If* $\mathcal{F}(\mathbf{x}) = \mathcal{F}'(\mathbf{x})\quad \forall \mathbf{x} \in X$*, then both UHCIs are the same piecewise* C^1 *function. Thus, we have* $Alg_1(\mathcal{T},\nu,\mathcal{U}) = Alg_1(\mathcal{T}',\nu',\mathcal{U}')$*.*

5.1.3 Construction of the Hierarchy from the Set of Separation Frontiers

Assume that we are given the set of separation frontiers $\mathcal{H}$ between the C^1 functions. We would like to be able to recover the hierarchy from this.

The idea is that, from Lemma 5, each term $\min_{i\in A} a_i$ (with $A \subseteq \mathrm{Ch}(k)$) in the expression of $\mathcal{A}_k$ yields the separation frontier $a_l = {a_{l'}}^1$ for any $\{l, l'\} \subseteq A$. In other words, the positive and negative values of the weights correspond to two separate sub-trees in the tree. Given the separation frontiers $\mathcal{H}$ (e.g. $\frac{{x_1}^2+{x_2}^3}{2} - \sqrt{x_4} = 0$ in

[1]Terms a_l and $a_{l'}$ can be moved independently so that we can reach $a_l < a_{l'}$ and $a_l > a_{l'}$. For instance, we can have $a_l = 0, 1$ or $a_{l'} = 0, 1$.

Function Partition$(N, \mathcal{K})$:
$\mathcal{R} = \{(i,j) \in N \times N, i \neq j \ : \ \exists (K^+, K^-) \in \mathcal{K},$
$\quad [\{i,j\} \subseteq K^+ \text{ or } \{i,j\} \subseteq K^-]\}$
$\{N_1, \ldots, N_q\} =$ connected components of $\langle N, \mathcal{R} \rangle$
For $l \in \{1, \ldots, q\}$
$\quad$ **If** $|N_l| > 1$
$\quad\quad \mathcal{K}_l = \{(K^+, K^-) \in \mathcal{K} \ : \ K^+ \cup K^- \subseteq N_l\}$
$\quad\quad \mathcal{N}_l =$ Partition$(N_l, \mathcal{K}_l)$
$\quad$ **Else**
$\quad\quad \mathcal{N}_l = \{N_l\}$
Return $(\mathcal{N}_1, \ldots, \mathcal{N}_q)$

Function $\text{Alg}_2(\mathcal{H})$:
Compute $\mathcal{K}$ from $\mathcal{H}$
Return Partition$(N, \mathcal{K})$

Algorithm 2: Obtaining the hierarchy from $\mathcal{H}$. N is either given by the problem, or trivially retrieved from $\mathcal{H}$.

Example 27), the idea is that the set of nodes having positive weights (e.g. $\{1,2\}$ in the Example) and the set of nodes having negative weights (e.g. $\{4\}$ in the Example) correspond to separate subsets of criteria in the tree (e.g. $\{1,2\}$ and $\{4\}$ belong to separate branches in Figure 5.2).

We consider the set of separation frontiers of the 2nd form and we look at the indices of the positive and negative weights – see Lemma 5:

$$\mathcal{K} = \Big\{ (K^+, K^-) : \{\mathbf{x} \in X : \sum_{l \in K} w_l \, u_l(x_l) = 0\} \in \mathcal{H}, \\ K^+ = \{l \in K, w_l > 0\} \text{ and } K^- = \{l \in K, w_l < 0\} \Big\}.$$

Algorithm 2 provides the construction of the tree $\mathcal{T}$ only from $\mathcal{K}$, following the previous idea that K^+ and K^- shall belong to separate parts of the tree.

We assume we are given a UHCI model $\mathcal{F}$, from which we know the separation frontiers $\mathcal{H}$. The following example illustrates the reconstruction of the hierarchy from $\mathcal{H}$. The correctness of Alg_2 directly derives from Lemma 5.

Example 28. *Let us consider a UHCI model over four attributes* $1, 2, 3, 4$*, with*

the following separation frontiers:

$$\mathcal{H} = \Big\{ {x_1}^2 + \frac{3{x_2}^3}{7} = \frac{10\sqrt{x_4}}{7}; \quad {x_1}^2 + \frac{3{x_3}^4}{7} = \frac{10\sqrt{x_4}}{7};$$
$${x_1}^2 + \frac{3{x_2}^3}{7} = \frac{20\,x_4 + 10}{21}; \quad {x_1}^2 + \frac{3{x_3}^4}{7} = \frac{20\,x_4 + 10}{21};$$
$${x_1}^2 = {x_3}^4; \quad {x_2}^3 = {x_3}^4; \quad x_4 = \frac{1}{4};$$
$${x_1}^2 + \frac{{x_2}^3}{2} + \frac{{x_3}^4}{6} = \frac{5\sqrt{x_4}}{3}; \quad {x_1}^2 + \frac{2{x_3}^4}{3} = \frac{5\sqrt{x_4}}{3};$$
$${x_1}^2 + \frac{{x_2}^3}{2} + \frac{{x_3}^4}{6} = \frac{10x_4+5}{9}; {x_1}^2 + \frac{2{x_3}^4}{3} = \frac{10x_4+5}{9} \Big\}.$$

We apply Alg_2 *to* $\mathcal{H}$*. We compute* $\mathcal{K} = \Big\{(\{1,2\},\{4\}),\ (\{1,3\},\{4\}),\ (\{1\},\{3\}),\ (\{1,2,3\},\{4\},\ (\{1,3\},\{4\}),\ (\{2\},\{3\})\Big\}$*. Then we compute the partitions:*

- Partition$(N,\mathcal{K})$*:* $\mathcal{R} = \{(1,2),(1,3),(2,3)\}$*. Hence the connected components of* $\langle N,\mathcal{R}\rangle$ *are* $\{1,2,3\}$ *and* $\{4\}$*. For the first set, we have* $\mathcal{K}_1 = \Big\{(\{1\},\{3\}),\ (\{2\},\{3\})\Big\}$.
- Partition$(\{1,2,3\},\mathcal{K}_1)$*:* $\mathcal{R} = \emptyset$*. Hence the connected components of* $\langle\{1,2,3\},\emptyset\rangle$ *are* $\{1\}$*,* $\{2\}$ *and* $\{3\}$*.*

The algorithm thus returns the tree of Figure 5.3.

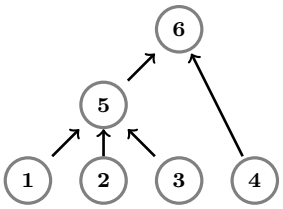


Figure 5.3: Tree constructed by Algorithm 2 from the separation frontiers from example 28.

5.1.4 Main Result

In order to get uniqueness properties, we need to introduce some assumptions on the HCI model.

Assumption H1: At every aggregation node $k \in V$, $\mathrm{Ch}(k)$ is the only connected component of graph $\langle \mathrm{Ch}(k), \{(i,j)\ ,\ i \neq j \text{ s.t. } \exists S \in \mathcal{S}_k\ :\ \{i,j\} \subseteq S\}\rangle$

H1 forbids to have a model C_{μ_k} that is (even only partly) additive. For instance, $a_5 = C_{\mu_k}(a_1, a_2, a_3, a_4) = \frac{1}{2}\min(a_1, a_2) + \frac{1}{2}\min(a_3, a_4)$ (see Figure 5.4-Left) violates H1 as groups $\{1, 2\}$ and $\{3, 4\}$ of variables are disconnected. In this example, we could obtain the same function with the tree of Figure 5.4-Right having two new aggregation nodes: $a_6 = \min(a_1, a_2)$, $a_7 = \min(a_3, a_4)$ and thus $a_5 = \frac{a_6+a_7}{2}$. Hence the hierarchy is clearly not unique in this example. On the other hand, $C_{\mu_k}(a_1, a_2, a_3, a_4) = \frac{1}{3}\min(a_1, a_2) + \frac{1}{3}\min(a_2, a_3) + \frac{1}{3}\min(a_3, a_4)$ satisfies H1 as the four variables are connected, and one cannot decompose C_{μ_k} with sub-aggregation nodes.

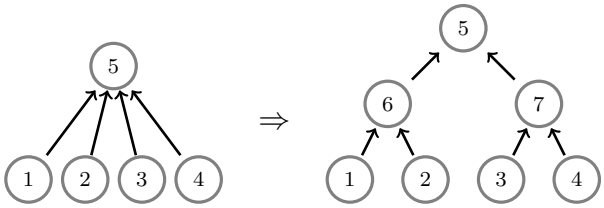


Figure 5.4: Illustration of H1.

Assumption H2: For all nodes $k \in V$:

$$|\mathcal{S}_k| \geq 2. \tag{5.3}$$

Assumption H2 (combined with H1) forbids from having a simple min between two variables. Intuitively, a simple min between two variables can be collapsed at the higher level. Consider the example of Figure 5.5-left, where $a_4 = \min(a_1, a_2)$ (violating H2) and $a_5 = \frac{a_3}{2} + \frac{\min(a_3,a_4)}{2}$. Then we can remove node 4 and directly write a_5 in terms on a_1, a_2, a_3: $a_5 = \frac{a_3}{2} + \frac{\min(a_1,a_2,a_3)}{2}$, which is a valid Choquet integral. Hence we can also represent this model with the tree of Figure 5.5-right.

The following lemma shows that under H1 and H2, applying successively Algorithm 2 on the set of separation frontiers produced by Algorithm 1 yields the same tree.

Lemma 7. *Let $\mathcal{F}$ a UHCI on tree $\mathcal{T} = \langle r, M, \mathrm{Ch}\rangle$, with a set of FM $\nu = \{\mu_k, k \in V\}$ and marginal utility functions $\mathcal{U} = \{u_1, \ldots, u_n\}$. If $\mathcal{F}$ satisfies* (4.4)*, $\mathcal{T}$ and ν satisfy H1 and H2, and $\mathcal{U}$ satisfies constraints* (4.1)*,* (4.2)*,* (4.3) *and* (5.2)*, then $Alg_2(Alg_1(\mathcal{T}, \nu, \mathcal{U})) = \mathcal{T}$.*

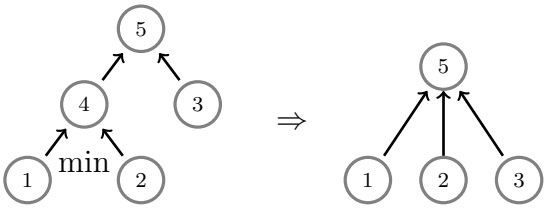


Figure 5.5: Illustration of H2.

The next result shows that the hierarchy of a UHCI model can be uniquely constructed from $\mathcal{F}$.

Theorem 2. *Under Assumptions H1, H2, and constraints* (4.1), (4.2), (4.3), (4.4) *and* (5.2), *there is a single tree that represents a UHCI model.*

Combining Theorems 1 and 2, we obtain our main result.

Theorem 3. *Let $\mathcal{F}$ and $\mathcal{F}'$ be two UHCI satisfying constraints* (4.5), *with potentially different hierarchies, fuzzy measures and marginal utility functions. Assume that both models fulfill H1, H2, and consrtaints* (4.1), (4.2), (4.3), (4.4), (4.5) *and* (5.2). *Then, both models have the same hierarchy, fuzzy measures and marginal utilities.*

5.2 Conclusion

We have, in this chapter, shown that, given a general UHCI, there is a unique possible parameterization that fits the assumptions presented in this chapter. The proofs of all lemmas and theorems evoked are given thereafter in Section 5.3. It is interesting to notice that the necessary conditions are very easily understandable, and make sense to a human DM; they hence do not conflict with the readability of the model. A possible extension would be, in the cases where either H1 or H2 is not met, to automatically build the unique equivalent model that meets both of these constraints.

This global unicity results allows to expect being able to trust the interpretation of a given model. This, in turn, is crucial for applications in safety-critical contexts. Moreover, interpretability is a very interesting result in machine learning. We thus present, in the next part of this thesis, how we leverage neural networks for learning these very same UHCI models.

5.3 Proofs

Lemma 8. *For all linear function of the marginal utilities* $\sum_{j\in\mathcal{S}(k)} w_j^k\, u_j(x_j) \in \mathcal{G}_k$, *where* $\mathcal{S}(k) \subseteq \mathrm{Lf}(k)$, *we have* $w_j^k > 0$ *for every* $j \in \mathcal{S}(k)$.

Proof : If there exists $j \in \mathcal{S}(k)$ with $w_j^k < 0$, then the score a_k would be locally decreasing w.r.t. this variable, which contradicts monotonicity of $\mathcal{A}$ w.r.t. its inputs. □

Proof of Lemma 5: The result is clear for $i \in N$.

For $k \in V$, the separation frontiers in $\mathcal{H}_k$ corresponds to an equation $g_i(x) = g_j(x)$. Nodes i and j corresponds to two distinct sub-trees with roots k' and k'' respectively. Hence by Lemma 8, all coefficients of g_i and g_j are positive. Hence the result. □

Proof of Lemma 6: The proof is done by induction. For leaves, the separation frontiers are the points of separation of the piecewise C^1 segments of the marginal utility functions. For the other nodes, the recursive construction of $\mathcal{G}_k$ and $\mathcal{H}_k$ is clear by Lemma 5. There is no required property on the HCI model to have this result. □

Notation: Set $\mathcal{G}_k$ is written as a set of linear models of the marginal utility functions. Each $g \in \mathcal{G}_k$ has a support, i.e. a set of leaves with non-zero weights; we call $\mathcal{T}_k$ the set of the supports of all $g \in \mathcal{G}_k$.

$$\mathcal{G}_k = \Big\{ \sum_{l\in T} w_l^T\, u_l(x_l)\ ,\ T \in \mathcal{T}_k \Big\}. \tag{5.4}$$

We denote by $\mathcal{G}'_k$ the set of linear models of the marginal utility functions of a_k as a function of $\mathbf{a}_{\mathrm{Ch}(k)}$.

Lemma 9. *Under assumptions (4.4) and H2, for every* $k \in M$, *the graph*
$\langle \mathrm{Lf}(k), \{(i,j),\ i,j \in T \text{ with } T \in \mathcal{T}_k\}\rangle$ *is connected, where set* $\mathcal{T}_k$ *in* $\mathcal{G}_k$ *is given by Eq. (5.4).*

Proof : The proof is done by backward induction.

For a leaf $k \in N$: $\mathcal{G}_k$ is the set of C^1 expressions that a_k can take, $\mathcal{H}_k$ is the set of $x_k = \theta$ for all θ where function u_k has a discontinuity of its derivative, and thus the assumption is proven at this node.

For an aggregation node $k \in V$: Let $\ell, \ell' \in \mathrm{Ch}(k)$ with $\ell \neq \ell'$. We need to show that ℓ, ℓ' are connected in the graph $Gr := \langle \mathrm{Ch}(k), \{(i,j),\ i,j \in T \text{ with } T \in \mathcal{T}'_k\}\rangle$ where $\mathcal{T}'_k$ is the set of coalitions appearing in $\mathcal{G}'_k$. By Assumption H2 and the non degeneracy of the capacities (see Lemma 1 derived from (4.4)), we have

$|\mathcal{S}_k| \geq 2$ and there exist $S_1, S_2 \in \mathcal{S}_k$ such that $\ell, \ell' \in S_1 \cup S_2$ and $S_1 \neq S_2$. The CI $C_{\mu_k}(\mathbf{a}_{\mathrm{Ch}(k)})$ at node k takes the form

$$m_k(S_1) \min_{i \in S_1} a_i + m_k(S_2) \min_{i \in S_2} a_i + \cdots \tag{5.5}$$

Two cases (the other cases are obtained by symmetry):

- 1st case: $\ell \in S_1$, $\ell' \in S_2$ and $\ell \notin S_2$. Hence by an appropriate ordering of the utilities, (5.5) can take the form $m_k(S_1)\, a_\ell + m_k(S_2)\, a_{\ell'} + \cdots$, so that ℓ and ℓ' are directly connected in Gr.
- 2nd case: $\{\ell, \ell'\} \subseteq S_1$ and $\exists \ell'' \in S_2$ with $\ell'' \notin S_1$. Hence by an appropriate ordering of the utilities, (5.5) can take the two forms $m_k(S_1)\, a_\ell + m_k(S_2)\, a_{\ell''} + \cdots$ and $m_k(S_1)\, a_{\ell'} + m_k(S_2)\, a_{\ell''} + \cdots$. Therefore ℓ and ℓ' are directly connected in Gr through ℓ''.

Replacing in the linear equation w.r.t. marginal utilities of $\mathcal{G}'_k$, terms a_ℓ ($\ell \in \mathrm{Ch}(k)$) by any element of $\mathcal{G}_\ell$, we obtain $\mathcal{G}_k$ and we easily see, by the induction assumption, that any pair $\ell, \ell' \in \mathrm{Lf}(k)$ is connected in $\mathcal{G}_k$. □

Proof of Lemma 7: Consider a UHCI model $\mathcal{F} = \langle \mathcal{T}, \nu, \mathcal{U} \rangle$, and let us apply Algorithms 1 and 2. Let us show that the hierarchy obtained by $\mathrm{Alg}_2(\mathrm{Alg}_1(\mathcal{F}))$ is exactly $\mathcal{T}$. More precisely, we show by induction starting from root down to the leaves that Algorithm 2 progressively produces hierarchy $\mathcal{T}$. We use below the sets $\mathcal{T}_\ell$ as defined in Eq. (5.4).

At the root node $k = r$, $\mathcal{H}_k$ is composed of the separation frontiers $\sum_{\ell \in T} w_\ell^T a_\ell = \sum_{\ell \in T'} w_\ell^{T'} a_\ell$ for every $T \in \mathcal{T}_\ell$ and $T' \in \mathcal{T}_{\ell'}$, where $\{\ell, \ell'\} \in \mathcal{S}_k$. In Algorithm 2, such a pair (T, T') is an element of $\mathcal{K}$. The leaves of two separate subtrees $\mathrm{Lf}(\ell)$ and $\mathrm{Lf}(\ell')$ are clearly separated.

Let $\ell \in \mathrm{Ch}(k)$. By H1, leaf ℓ is necessarily connected to another node ℓ' through $\mathcal{S}_k$. Hence, a separation frontier at node k takes the form $a_\ell = a_{\ell'}$. By Lemma 9, all leaves are connected through $\langle \mathrm{Lf}(k), \{(i,j),\ i,j \in T \text{ with } T \in \mathcal{T}_k\} \rangle$. Therefore all leaves of a child $\ell \in \mathrm{Ch}(k)$ form a connected component of $\langle \mathrm{Lf}(k), \mathcal{R} \rangle$. Hence we obtain the right grouping of the elementary criteria at the top level.

We finally reproduce the previous reasoning recursively on any aggregation node k. □

Proof of Theorem 2: Consider two UHCI models $\mathcal{F} = \langle \mathcal{T}, \nu, \mathcal{U} \rangle$ and $\mathcal{F}' = \langle \mathcal{T}', \nu', \mathcal{U}' \rangle$ that both satisfy Assumptions H1, H2, (4.1), (4.2), (4.3), (4.4), (4.5) and (5.2). Assume that these models are different and in particular $\mathcal{T} \neq \mathcal{T}'$.

Assume by contradiction that these two UHCI models yield exactly the same overall utility model. By Lemma 6, Algorithm 1 to $\mathcal{F}$ and $\mathcal{F}'$ yields the piecewise C^1 functions and separating frontiers underlying $\mathcal{F}$ and $\mathcal{F}'$. By Corollary 1 of

Lemma 6, set $\mathcal{H}$ obtained by Algorithm 1 is identical. That is $\text{Alg}_1(\mathcal{T},\nu,\mathcal{U}) = \text{Alg}_1(\mathcal{T}',\nu',\mathcal{U}) = \mathcal{H}$.

This equality yields that $\text{Alg}_2(\text{Alg}_1(\mathcal{T},\nu,\mathcal{U})) = \text{Alg}_2(\text{Alg}_1(\mathcal{T}',\nu',\mathcal{U}')) = \mathcal{T}''$. Lemma 7 says that $\mathcal{T} = \mathcal{T}''$ and $\mathcal{T}' = \mathcal{T}''$, as both $\mathcal{F}$ and $\mathcal{F}'$ satisfy H1, H2, (4.1), (4.2), (4.3), (4.4), (4.5) and (5.2). We conclude that $\mathcal{T}$ and $\mathcal{T}'$ are identical. We raise a contradiction. □

Proof of Theorem 3: Consider two UHCI models $\mathcal{F} = \langle \mathcal{T},\nu,\mathcal{U}\rangle$ and $\mathcal{F}' = \langle \mathcal{T}',\nu',\mathcal{U}'\rangle$ that both satisfy assumptions H1, H2, (4.1), (4.2), (4.3), (4.4), (4.5) and (5.2).

By Theorem 2, we have $\mathcal{T} = \mathcal{T}'$. As we are now in a setting where both UHCIs have the same hierarchy, we can apply Theorem 1, which gives us that $\nu = \nu'$ and $\mathcal{U} = \mathcal{U}'$, concluding the proof. □

Part IV

Neural Representation of MCDA Models

CHAPTER 6

MARGINAL UTILITY MODULES

Contents

6.1 Motivations

We have shown in Part III that the UHCI model is identifiable under mild conditions. This is an interesting result for interpretability, but also for learning such models from data.

This part will be dedicated to presenting the main technical contribution of this PhD: the NEUR-HCI framework. The ideas behind NEUR-HCI are:

- Learn a UHCI model from data (both marginal utilities and the FM of each aggregator)
- Have formal guarantees that the learned model be a valid UHCI (i.e. it meets all of the constraints needed)
- All of this from data

The main idea is that the thus learned model be a valid UHCI. As such, NEUR-HCI exploits the qualities of both MCDA (in terms of formal constraints and interpretability of the model) and machine learning (using data instead of an expert decision maker for building the model). The approach that we chose is using so-called *neural modules*, which are small multilayer perceptrons. This approach comes from the highly modular structure of the UHCI, as the hierarchy is a graphical structure between several aggregators (CIs), with marginal utilities at the leaves. As a consequence, there will be two main types of modules in NEUR-HCI, each able to represent and learn a certain class of functions:

- marginal utility modules, whose implementations are presented in this chapter
  - monotonic (non-decreasing/non-increasing)
  - bitonic (single-peaked/single-valleyed)
- aggregation modules, whose implementation is presented in Chapter 7
  - 2-additive Choquet integrals
  - a subset of the 3-additive Choquet integrals

– general Choquet integrals

It is also interesting to see that the constraints of the UHCI model come from constraints that are enforced locally (i.e. if all modules are locally valid, then the whole model is). It thus seems relevant to leverage the efficiency of backpropagation for learning such models. As a consequence, we want each of our modules to be able to represent a dense part of their associated model class, so that our search-space is dense in the space of all possible UHCIs.

Moreover, this modular approach enables experts to validate, modify, or restructure the tree whenever they need (in which case it might be necessary to retrain the model, or at least parts of it). It also allows module-wise interpretation, and thus validation, by a domain expert. In term, there is run-time interpretability, which means that a DM using the model can at all time understand the output of the model for a given alternative or decision, making it relevant in safety-critical contexts.

It is to be noted that, once the model is trained, the neural network structure can be discarded, and the model re-written as a simple mathematical formula, making it highly compressible and fit for embedding in resource-scarce environments (i.e. FPGA boards).

The work presented in this part resulted in two papers, in IJCAI2020 [Bresson et al., 2020c] and DA2PL2020 [Bresson et al., 2020b].

We recall that we assume a marginal utility u_i to be of type t, among:

- non-decreasing (ND)
- non-increasing (NI)
- single-peaked (SP)
- single-valleyed (SV)

Let $\mathcal{E}_t$ the set of all functions of type $t = \{\text{ND}, \text{NI}, \text{SP}, \text{SV}\}$ which satisfy (4.1), (4.2) and (4.3). Then $\mathcal{E}_t$ is the set of all valid marginal utilities of type t.

We present in this chapter several architectures of multilayer perceptrons, which we call marginal utility modules. There are four types of such modules, one for each type of marginal utilities.

We want a marginal utility module Q of type t to satisfy the following requirements:

- **Representativity**: be able to approximate any $f \in \mathcal{E}_t$ with an arbitrary precision;

- **Constrainedness**: only be able to represent a valid marginal utility of type t (that is: $\mathrm{Q} \in \mathcal{E}_t$)

Formally, we want the set of all possible modules of type t to be a dense part of $\mathcal{E}_t$. Below, we write indifferently Q to describe the neural network and the function it represents.

In Section 6.2, we introduce the logistic sigmoid, which will be the building block of our parametric representations. Then, in Section 6.3, we show how we implement a monotonic marginal utility (either non-decreasing or non-increasing). Then, in Section 6.4, we show how we derive the bitonic ones (either single-peaked or single-valleyed). Finally, in Section 6.5, we describe a selector, that is, a module which can select the best type of marginal utilities during training.

In the remainder of this section, and for the sake of readability, we will drop the subscript "$_i$". u will thus be a marginal utility, and x the value on a given attribute.

6.2 Logistic Sigmoid

Definition 9. *A logistic sigmoid $\sigma_{\eta,\beta}$ is a real mathematical function, which can be written as:*

$$\begin{aligned} \sigma_{\eta,\beta} :& \mathbb{R} \to]0,1[\\ & x \mapsto \frac{1}{1+e^{-(\eta x-\beta)}} \end{aligned}$$

with η called the *steepness* of *precision* parameter, and β called the *bias*. The effect of η can be seen in Figure 6.1. Below, we write $\sigma = \sigma_{1,0}$ the standard logistical sigmoid centered on 0 with steepness 1. Obviously, we have $\sigma_{\eta,\beta}(x) = \sigma(\eta x - \beta)$.

One advantage of the logistic sigmoid is that its gradient is quickly computed from its value. That is:

$$\begin{aligned} \frac{\partial \sigma}{\partial x}(x) &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{e^{-x}}{(1+e^{-x})} \cdot \frac{1}{(1+e^{-x})} \\ &= \left(\frac{1+e^{-x}-1}{(1+e^{-x})}\right) \cdot \sigma(x) \\ &= (1-\sigma(x))\sigma(x) \end{aligned}$$

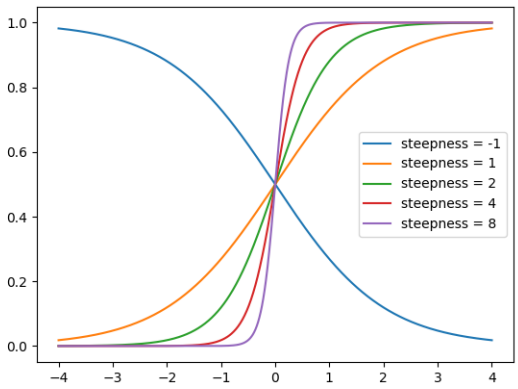


Figure 6.1: Sigmoids with varying steepness parameters; $\beta = 0$.

Thus, given $\sigma(g(x))$ with g a differentiable function, we have, by the chain rule:

$$\frac{\partial \sigma \circ g}{\partial x}(x) = (1 - \sigma(g(x)))\sigma(g(x))\frac{\partial g}{\partial x}(x)$$

In particular:

$$\frac{\partial \sigma_{\eta,\beta}}{\partial x}(x) = (1 - \sigma(g(x)))\sigma(g(x))\frac{\partial g}{\partial x}(x)$$

This ease of computation makes the sigmoid suitable for computing gradients for the backpropagation algorithm.

6.3 Monotonic Marginal Utility

6.3.1 Non-decreasing marginal utility

6.3.1.1 Parametric representation

We assume in this subsection that $t = \text{ND}$, *non-decreasing*.

In order to learn the functions in $\mathcal{E}_{\text{ND}}$, we need to be able to write them in a parametric form. Following the work in [Fallah Tehrani et al., 2014], we write a marginal utility $u \in \mathcal{E}_{\text{ND}}$ as a convex sum of logistic sigmoids:

$$u(x) = \sum_{k=1}^{h} r_k s_k(x) := \sum_{k=1}^{h} \frac{r_k}{1 + e^{-(\eta_k x - \beta_k)}} \tag{6.1}$$

By writing $r = (r_1, ..., r_h)$, $\beta = (\beta_1, ..., \beta_h)$ and $\eta = (\eta_1, ..., \eta_h)$, we can rewrite this as:

$$u(x) = r \bullet \sigma(\eta x - \beta)$$

with $\bullet$ the dot product, σ applied element-wise, and where the hyper-parameter h sets the maximum number of sigmoids involved in the representation[1] ; β_k and η_k respectively are the bias and precision parameters of the k-th sigmoid, and r_k its weight.

The fact that this sum is convex means that we impose the following conditions on the weight vector $r = (r_1, ...r_k)$.

$$\forall k \in \{1, \dots, h\} : r_k > 0 \text{ and } \sum_{k=1}^{h} r_k = 1 \tag{6.2}$$

We also impose one positivity condition on η_i:

$$\forall k \in \{1, \dots, h\} : \eta_k > 0 \tag{6.3}$$

Theorem 4. *Let $\mathcal{E}^*_{\text{ND}}$ be the set of all functions obtained by varying the weights, steepness, and biases in Equation* (6.1)*, under Constraints* (6.3) *and* (6.2)*.*

Then $\mathcal{E}^*_{\text{ND}}$ is a dense part of $\mathcal{E}_{\text{ND}}$.

Proof :

1. Proof that $\mathcal{E}^*_{\text{ND}} \subseteq \mathcal{E}_{\text{ND}}$:

Proving this inclusion is equivalent to showing that, $\forall u \in \mathcal{E}^*_{\text{ND}}$, u is non-decreasing, and u satisfies Constraints (4.1), (4.2) and (4.3). That is, it states that any function build according to Equation (6.1) is a valid non-decreasing marginal utility.

Non-decreasingness: by (6.3), all s_k in Equation (6.1) are non-decreasing. By Equation (6.2), $u \in \mathcal{E}^*_{\text{ND}}$ is a sum with non-negative coefficients of non-decreasing functions. It is thus, in turn, non-decreasing.

Normalization: by Equation (6.2), the r_k sum to 1. As, $\forall s_k$, $s_k(x) \xrightarrow[x \to \infty]{} 1$ and $s_k(x) \xrightarrow[x \to -\infty]{} 0$, we have the same limits for u by combination of the limits. Together with the non-decreasingness, this yields (4.1), (4.2) and (4.3).

This shows that any function written as in (6.1) under (6.3) and (6.2) is a valid non-decreasing marginal utility.

[1] The actual number of sigmoids is minimized through L_1 regularization, Eq. (8.1).

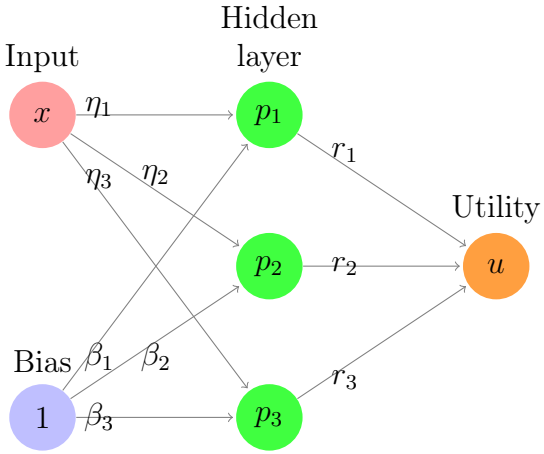


Figure 6.2: A utility module with 3 hidden nodes ($h = 3$).

2. Proof that $\mathcal{E}^*$ is dense in $\mathcal{E}$:

A variant of the universal approximation theorem [Daniels and Velikova, 2010] tells us that, given any sigmoids components, we can approximate any function in $\mathcal{E}_{\text{ND}}$ by a function of $\mathcal{E}^*_{\text{ND}}$. This representation is thus suitable for parameterizing the class of non-decreasing marginal utilities. □

6.3.1.2 Implementation as a neural module

Another advantage of the form (6.1) is that it is exactly a simple FNN with:

- 1 input neuron
- h hidden neurons $p_1, ..., p_h$ on a single layer with sigmoidal activations
- 1 output neuron

The neuron are classic linear neurons. There is thus a single weight (resp. bias) between the input neuron and hidden neuron p_j, and this weight (resp. bias) is η_j (resp. β_j). The weight between hidden neuron p_j and the output is r_j. There is no bias on the output layer. This is illustrated in Figure 6.2 with $h = 3$.

In this module, the forward phase is thus as follows:

1. the input neuron/layer receives a single number (or 1-d vector) x;
2. each hidden neuron p_j receives $\eta_j x - \beta_j$ as an input;
3. each hidden neuron returns its sigmoidal activation, that is $\sigma(\eta_j x - \beta_j)$;

4. the output of each hidden neuron p_j is weighed by the associated r_j;

5. the output node sums all of its inputs, yielding exactly the value from (6.1).

6.3.2 Computing the gradient

Now that we have seen how to represent and implement a non-decreasing marginal utility, it is necessary to be able to compute the gradients of the error, in order to readjust the parameters during training.

Below, by abuse of notation, we write $p_j = s_j(x)$ the output of node p_j, and u the output of the output node.

The parameters that have to be learned are the η_j, the β_j, and the r_j. We give here the explicit expression for the gradient of each parameter, given $\frac{\partial \text{err}}{\partial u}$.

6.3.2.1 Gradient for the r_j

$$\begin{aligned}\frac{\partial \text{err}}{\partial r_j} &= \frac{\partial \text{err}}{\partial u} \cdot \frac{\partial u}{\partial \beta_j} \frac{\partial u}{\partial r_j} \\ &= \frac{\partial \text{err}}{\partial u} \cdot p_j\end{aligned}$$

6.3.2.2 Gradient for the β_j

$$\begin{aligned}\frac{\partial \text{err}}{\beta_j} &= \frac{\partial \text{err}}{\partial u} \cdot \frac{\partial u}{\partial \beta_j} \\ &= \frac{\partial \text{err}}{\partial u} \cdot \frac{\partial u}{\partial p_j} \cdot \frac{\partial p_j}{\partial \beta_j} \\ &= \frac{\partial \text{err}}{\partial u} \cdot r_j \cdot p_j(1 - p_j)\end{aligned}$$

6.3.2.3 Gradient for the η_j

$$\begin{aligned}\frac{\partial \text{err}}{\eta_j} &= \frac{\partial \text{err}}{\partial u} \frac{\partial u}{\partial \eta_j} \\ &= \frac{\partial \text{err}}{\partial u} \cdot \frac{\partial u}{\partial p_j} \cdot \frac{\partial p_j}{\partial \eta_j} \\ &= \frac{\partial \text{err}}{\partial u} \cdot r_j \cdot p_j(1 - p_j)x\end{aligned}$$

6.3.2.4 Gradient for the x (for backpropagating to potential previous modules)

$$\begin{aligned}\frac{\partial \text{err}}{x} &= \frac{\partial \text{err}}{\partial u} \cdot \frac{\partial u}{\partial x} \\ &= \frac{\partial \text{err}}{\partial u} \cdot \sum_{j=1}^{h} \left(\frac{\partial u}{\partial p_j} \cdot \frac{\partial p_j}{\partial x} \right) \\ &= \frac{\partial \text{err}}{\partial u} \cdot \sum_{j=1}^{h} (r_j \cdot p_j(1-p_j)\eta_j)\end{aligned}$$

6.3.3 Ensuring the Validity of the Module

As stated in the previous sections, there are constraints to ensure on the parameters so that the model is valid. Namely, we need to satisfy (6.2) for r and (6.3) for η. By definition, during the gradient-based learning process, it might happen that the parameter start violating these constraints. We thus need some procedures to prevent this from happening.

The positivity constraints on r are ensured through clipping. That is, if r exits the subspace $\mathbb{R}_+^h$, by having negative values on some coordinates, we project it orthogonally back on $\mathbb{R}_+^h$ by setting those values to 0. The process is analogous for η.

For the normalization constraint in (6.2), we linearly re-normalize the parameters after each update; that is, we divide r by $\sum_{j=1}^{h} r_j$.

Note that these normalizations and clipping are also done right after initialization, so the module starts training from the valid search space. These procedures ensure that, at all time during training, the function represented by the module is a valid non-decreasing marginal utility. This validity is formally enforced by the architecture and constraints on the parameters, and cannot be violated, even locally.

6.3.4 Non-increasing marginal utility

Now that we have seen how to build a non-decreasing marginal utility, it is trivial to build a non-increasing one. There are actually two easy methods to do this:

First of all, we can initialize η to be non-positive, and clip it to zero on the dimensions where its values become positive. Nonetheless, for the sake of factorization, we favored an other approach: we represent our non-increasing u as $1 - \text{Q}$, where Q is a non-decreasing utility module. Note that this implies to multiply all of the gradients of Q by -1 during training.

Corollary 2 (of Theorem 4)**.** *We define* $\mathcal{E}^*_{\text{NI}}$ *as* $\mathcal{E}^*_{\text{NI}} := \{1-u, u \in \mathcal{E}^*_{\text{ND}}\}$*. Then* $\mathcal{E}^*_{\text{NI}}$ *is a dense part of* $\mathcal{E}_{\text{NI}}$*.*

6.4 Bitonic Marginal Utility

We have seen in the previous section how to represent and learn any monotonic marginal utility, while respecting the constraints set so that the model remains valid. In this section, we generalize the class of marginal utilities that we can represent to bitonic functions; that is, either single-peaked or single-valleyed.

In the remainder of this section, with the exception of Section 6.4.7, we assume that $t = \text{SP}$ is the single-peaked type. That is, it is non-decreasing on an interval $]-\infty, T[$, then non-increasing on $]T, +\infty[$, with $T \in \mathbb{R}$ called the *threshold*.

Note that T might not be uniquely-defined; there might be a so-called *plateau*, where the function remains constant and equal to 1 on an interval. Any value in this interval can be a candidate for T.

6.4.1 Validity Constraints

The fact that we are now working with bitonic marginal utilities does not change the normalization constraints (4.1), (4.2) and (4.3). Nonetheless, we add now a new constraint, which is that:

$$\lim_{x \to -\infty} u(x) = \lim_{x \to \infty} u(x) \tag{6.4}$$

This ensures that u has (potentially asymptotically) the extreme value 0 (resp. 1) on the bounds of the intervals if u is single-peaked (resp. single-valleyed).

Just like before, we want u, and any trained approximation, to be formally non-decreasing on its left part, that is, on interval $]-\infty, T[$, and formally non-increasing on its right part $]T, +\infty[$. A single-peaked function thus reaches 1 in T, and tends to 0 on the bounds on the interval.

6.4.2 Parametric Representation

We have seen in Section 6.3.1.1 how to parameterize a monotonic marginal utility. We use this in order to build a parametric version of the single-peaked marginal utility u. We only need three components:

- a threshold $T \in \mathbb{R}$
- a non-decreasing marginal utility Q, s.t. $\text{Q}(T) = 1$

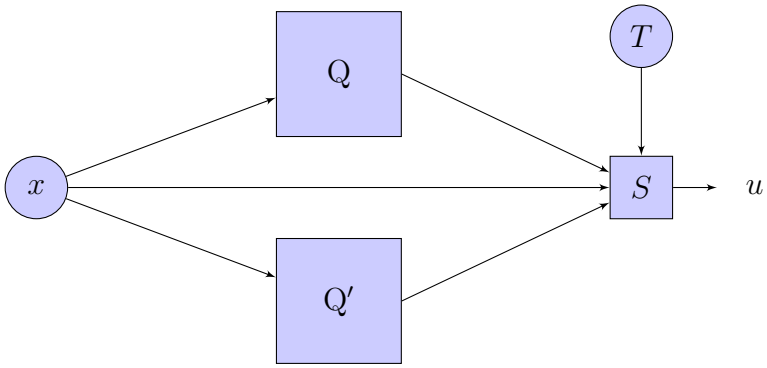


Figure 6.3: Bitonic single-peak utility module. S is a switch node, which returns $\mathrm{Q}(x)$ if $x < T$, $\mathrm{Q}'(x)$ otherwise.

- a non-increasing marginal utility Q', s.t. $\mathrm{Q}'(T) = 1$

Note that (m, m') must be in $(\mathcal{E}_{\mathrm{ND}} \times \mathcal{E}_{\mathrm{NI}})$. Then, we can compute u:

$$u(x) = \begin{cases} \mathrm{Q}(x) & \text{if } x \leq T \\ \mathrm{Q}'(x) & \text{if } x \geq T \end{cases}$$

As we want u to be continuous, note that we have:

$$\mathrm{Q}(T) = \mathrm{Q}'(T) = 1 \tag{6.5}$$

This implies that Q and Q' must reach 1 in a finite value, and not asymptotically.

6.4.3 Implementation as a Neural Module

A single-peaked marginal utility module $\tilde{\mathrm{Q}}$ will thus be implemented as a composite network, composed of:

- a threshold parameter T
- a non-decreasing marginal utility module Q
- a non-increasing marginal utility module Q'
- a switch node S

This is illustrated in Figure 6.3. The switch node S takes $x,\ T,\ \mathrm{Q}(x),\ \mathrm{Q}'(x)$ as inputs. It returns $\mathrm{Q}(x)$ if $x \leq T$, $\mathrm{Q}'(x)$ otherwise.

The set of functions representable by such modules is exactly $\mathcal{E}^*_{\mathrm{SP}}$, which is a dense part of the set we wish to learn.

6.4.4 Computing the Gradient

For training the module, we need to compute the gradient for each parameter. The parameters are:

- all of the internal parameters of Q and Q′ as monotonic modules, that is, their respective r, β and η, as described in section 6.3
- the threshold T

6.4.4.1 Gradient for the Parameters of the Monotonic Components

The first point has already been described in 6.3. As Q and Q′ are monotonic marginal utility modules, we can compute their gradient as presented. The nuance is that they need to be applied only for the module that was used for forward-propagation; that is, given $\theta \in \{r, \eta, \beta\}$ (resp. $\theta \in \{r', \eta', \beta'\}$) a parameter of Q (resp. Q′), the gradient is:

$$\frac{\partial \text{err}}{\partial \theta^j} = \mathbb{1}_{]-\infty,T[}(x) \left(\frac{\partial \text{err}}{\partial u} \cdot \frac{\partial u}{\partial \theta^j} \right)$$

In the same way, for $\theta' \in \{r', \eta', \beta'\}$) a parameter of Q′, we have:

$$\frac{\partial \text{err}}{\partial \theta^j} = \mathbb{1}_{[T,-\infty[}(x) \left(\frac{\partial \text{err}}{\partial u} \cdot \frac{\partial u}{\partial \theta^j} \right)$$

Where $\mathbb{1}_{\{S\}}$ is the characteristic function of set S; that is, $\mathbb{1}_{\{S\}} = 1$ if $x \in S$, 0 otherwise.

6.4.4.2 Gradient of the Threshold Parameter

Computing the gradient for T is mode complex. Indeed, u is not necessarily differentiable in T. Thus, we need to compute a surrogate, or sub-gradient, to have an update rule for T. After testing, we settled for the following surrogate:

$$\delta_t(x) = (x - t) \tag{6.6}$$

The stochastic update rule thus becomes:

$$T \leftarrow T - \rho_T \delta_T(x) \frac{\partial \text{err}(x)}{\partial u} \tag{6.7}$$

with ρ_t the learning rate for t.

6.4.5 Enforcing the Constraints on the Network

Let $\tilde{\mathrm{Q}} = \langle M, \mathrm{Q}', T \rangle$ a single-plateau utility module implemented as described in the above sections. In order for it to be valid, we need it to satisfy (4.1), (4.2), (4.3) and (6.4). We have seen that this is equivalent to having:

1. (4.1), (4.2) and (4.3) for both Q and Q′
2. $\lim_{x \to \infty} \tilde{\mathrm{Q}}(x) = \lim_{x \to -\infty} \tilde{\mathrm{Q}}(x) = 0$
3. $\tilde{\mathrm{Q}}(T) = 1$

Points 1. and 2. are taken care of through the same methods than presented in 6.3.3. They ensure the validity of both Q and Q′ as monotonic marginal utility functions of their respective types, and thus yields the right constraints.

Point 3., nonetheless, is a little trickier. Indeed, the representation given in (6.1) does not allow to reach 1; it is only reached asymptotically.

In order to solve this problem, we use another type of renormalization. The procedure is the same for Q and Q′, we describe it only for Q here.

First of all, we apply the procedures that we have already evoked, to normalize Q as a valid monotonic marginal utility. After this point, we have $\sum_{i=j}^{p} r_j = 1$. Then, we have $0 < \mathrm{Q}(T) < 1$.

We then apply the following transformation on r_j:

$$r \leftarrow \frac{r}{\mathrm{Q}(T)} \tag{6.8}$$

Considering that the expression in (6.1) is linear in r, applying this procedure to both Q and Q′ ensures that we have $\mathrm{Q}(T) = \mathrm{Q}'(T) = 1$. Note that, after this operation, it is no longer true that $\forall x \in \mathbb{R},\ \mathrm{Q}(x) \leq 1$. In particular, $\forall x > T,\ \mathrm{Q}(x) \geq 1$. Nonetheless, due to the switch node, we know that $\mathrm{Q}(x)$ will never be returned for any value of x that is not in $]-\infty, T]$. On the other hand, the fact that $\lim_{x \to \infty} \mathrm{Q}(x) = 0$ has been left unchanged.

The same considerations can be made for Q′.

As a consequence, after this renormalization, all of the constraints evoked above are satisfied. If we apply it after normalizing Q and Q′ individually, at each epoch or parameter update, we ensure that $\tilde{\mathrm{Q}}$ satisfies all constraints that are necessary for its validity as a single-peaked marginal utility.

In order to tackle some empirical instabilities, with exploding values for the r_k, we also use a second type of parameters re-adjustments. Indeed, if T gets small, Q is only exploited on a small portion of the unit interval, namely $[0, T]$. As a

consequence, all of the sigmoids s_j with a precision parameter $\beta_j >> T$ will only contribute marginally to the modules values on $[0, T]$ (i.e. $s_j(T) \sim 0$). This leads to small values of $\mathrm{Q}(T)$; thus, the transformation presented in Equation (6.8) leads to exploding r.

In order to compensate for that, we simply instore a rule to eliminate the sigmoids whose role is negligeable. That is, we set to 0 the weight of any sigmoid whose bias parameter is above a certain threshold. Namely:

$$r_j \leftarrow \begin{cases} r_j & \text{if } \beta_j < T \\ 0 & \text{otherwise} \end{cases}$$

For large value of T, the symetric procedure is applied to Q'.

6.4.6 Re-characterization

It is to be noted that a monotonic marginal utility is a special case of a single-peaked marginal utility. In particular, if T tends to infinity, $\tilde{\mathrm{Q}}$ is a non-decreasing function. On the other hand, if T tends to $-\infty$, $\tilde{\mathrm{Q}}$ is non-increasing.

Thus, it seems logical to have a criterion which would allow to detect whether $\tilde{\mathrm{Q}}$ is trying to learn a monotonic function (that is, if the underlying distribution requires a monotonic relation). This decision criterion can then be used to identify the underlying function, and change the type of marginal utility module accordingly. This would allow to lighten the training burden, as we now have to train only a single monotonic module to train, rather than two.

Note that re-characterization is part of the training; that is, it can be done entirely during the training step, without interrupting it. Nonetheless, it might be relevant to make sure the model trains for sufficient epochs after re-characterization, to ensure that the model parameters adapt to this quite brutal change.

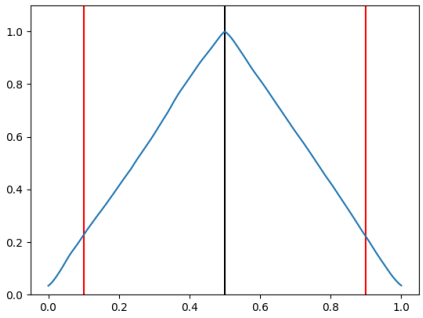
We have retained two recharacterization criteria:

6.4.6.1 Threshold-based

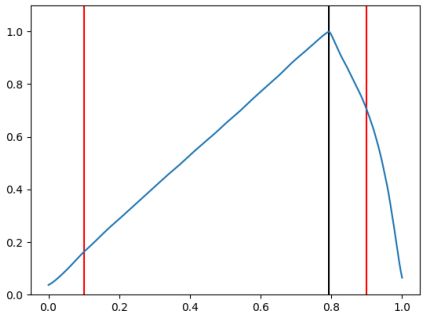
The method presented here is illustrated in Figure 6.4.

We need three parameters for this criterion: two extreme values $x_1^* < x_2^* \in \mathbb{R}$ and a number of epochs $k \in \mathbb{N}$. The criterion works as follows: if we reach k consecutive epochs during which $T < x_1^*$ (resp. $T > x_2^*$), then we replace $\tilde{\mathrm{Q}}$ by Q' (resp. Q).

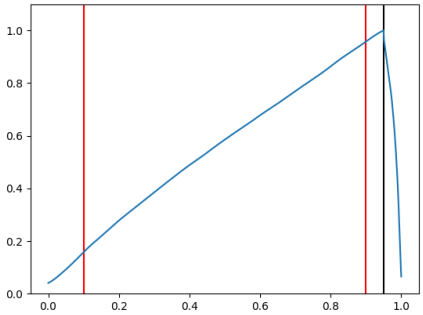
The extreme value parameters are usually chosen such that there is little to no data points with a value outside of $[x_1^*, x_2^*]$ on the attribute associated with the marginal utility, such that these values are considered extreme cases of the observed data (in the sense that there is low probability that any value more extreme can be observed). For instance, if the data was standardized to variance



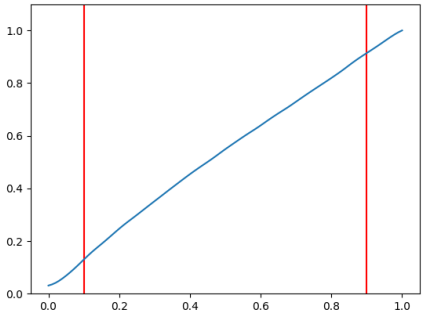
(a) At initialization, $T = 0.5$



(b) After some epochs, T goes towards 1



(c) $T > 0.9$, we start counting the epochs during which it remains there



(d) T has remained greater than 0.9 for more than k epochs: $\tilde{Q}$ was re-characterized as a non-decreasing function.

Figure 6.4: Threshold-based re-characterization process of a single-peaked marginal utility module into a non-decreasing one: the red vertical lines show the extreme values parameters 0.1 and 0.9, the vertical black line shows the current threshold.

1 and mean 0, as is usual, we could chose $-x_1^* = -1.96$ and $x_2^* = 1.96$, as these are the bounds of the confidence interval of 95% for such a standardized normal distribution. Field knowledge might also provide such values. Otherwise, if the data was normalized linearly on the unit interval, one might use $x_1^* = \epsilon$, $x_2^* = 1 - \epsilon$ for a given $\epsilon > 0$.

The k parameter gives the tolerance that the model has w.r.t. re-characterization. The higher k, the more the model will need confirmation before making its decision, letting time for T to re-enter interval $[x_1^*, x_2^*]$. The smaller k, the quicker such decision will be taken. These are hyperparameters of the model; in practice, the training is quite sensitive to them, it is thus important to tune them carefully.

6.4.6.2 Extrema-based

The method presented here is illustrated in Figure 6.5.

We need four parameters for this criterion: two extreme values $x_1^* < x_2^* \in \mathbb{R}$, one tolerance parameter t and a number of epochs $k \in \mathbb{N}$. The criterion works as follows: if we reach k consecutives epochs during which $\tilde{\mathrm{Q}}(x_1^*) > t$ (resp. $\tilde{\mathrm{Q}}(x_0^*) > t$), we replace $\tilde{\mathrm{Q}}$ by Q' (resp. Q).

Informally, this means that if the non-decreasing (resp. non-increasing) part of the module is "not increasing enough" (resp. "not decreasing enough"). Thus, we eliminate this part and conserve only the other one.

As before, the extreme values should be chosen such that there is little to no observed data outside of the interval $[x_1^*, x_2^*]$, in the same way as before. The t parameter will need to be tuned according to the noise in the data, and the tolerance we want the model to have. For non-noisy data, some values as low as 0.1 can be used; on the other hand, using a low t on noisy data will make the model more prone to re-characterizing. This can be mitigated by using a larger k, which will require the model to stabilize in the "critical zone" (where $\tilde{\mathrm{Q}}(x_1^*) > t$ or $\tilde{\mathrm{Q}}(x_2^*) > t$) for a longer time in order to make its decision.

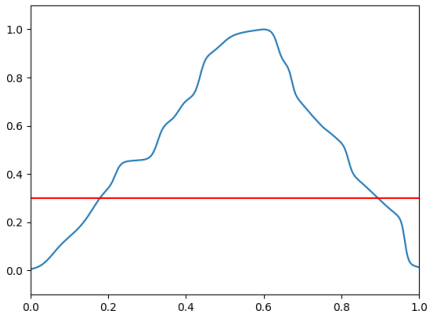
Finally, it is obvious that the learning rate plays an important role as well. Indeed, if it is too small, the model might take a large number of epochs in order to escape the critical zone, thus making the model more prone to re-characterizing. On the other hand, to large a learning rate might make the model jump in and out of said critical zone, and it will never stabilize in or out of it, leading to no re-characterization.

6.4.6.3 Precise procedure for re-characterization

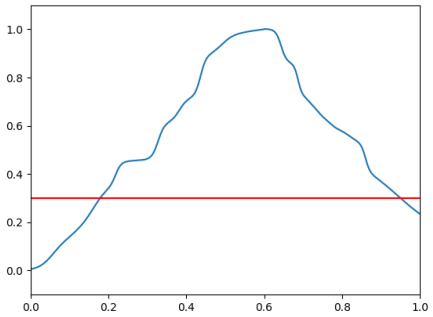
Once the chosen criterion for re-characterization has been fulfilled, the module $\tilde{\mathrm{Q}}$ can be replaced by its selected component, that is, either Q or Q', which is then normalized so that it fits the constraints for a monotonic marginal utility. We assume, without loss of generality, that the non-decreasing part Q was the one selected.

Then, several options are possible:

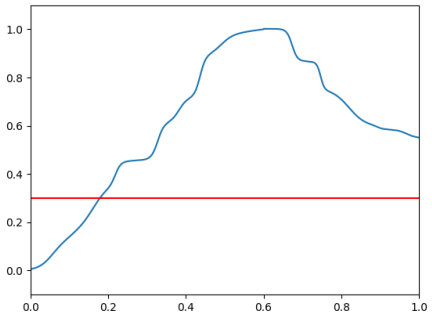
- The first one involves resetting the network, but using the used type of Q (namely non-decreasing) in place of the former $\tilde{\mathrm{Q}}$, then training again. In this case, the only information extracted from the previous training is the type of Q. This option is the safest, as it ensures that all marginal utilities have the same training epochs, and that the error on the type of $\tilde{\mathrm{Q}}$ (which was single-peaked rather than non-decreasing as it should have been) will not influence the training of the other parameters. Thus, we lose any information learned before. This is conceivable if there are a limited number



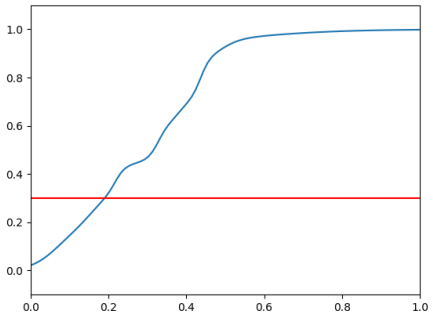
(a) At initialization, $\tilde{Q}$ represents a valid single-peaked marginal utility



(b) After some epochs, $Q'(1)$ starts going up, but remains in the tolerance interval (between 0 and 0.3, represented by a horizontal line)



(c) $Q'(1) > 0.3$, we start counting the epochs during which it remains there



(d) $Q'(1)$ has remained greater than 0.9 for more than k epochs: $\tilde{Q}$ was re-characterized as a non-decreasing function.

Figure 6.5: Extrema-based re-characterization process of a single-peaked marginal utility module into a non-decreasing one: the red horizontal line shows the tolerance parameter 0.3; we have also $x_1^* = 0$ and $x_2^* = 1$

of re-characterizations in the training, but it can quickly make training much longer, as the model is reset each time a re-characterization occurs.

- A more efficient procedure is to keep the whole network as is, with the exception that $\tilde{\mathrm{Q}}$ is replaced with a freshly initialized non-decreasing module Q''. This ensures that the other parameters retain the memory of their previous training, and do not start from scratch when running new training epochs. Nonetheless, it also makes Q'' late in terms of training, as it is initialized while the other modules in the network have already greatly progressed in terms of optimization. If we are using decreasing learning rate, for instance, this means that all other modules require a small learning rate for fine tuning, while Q'' requires a larger one for starting the whole initialization process from scratch.

- Finally, the third method involves keeping all of the information learned, both on the network and on Q. In this case, we use Q as is, potentially re-normalizing it in order to make it a valid monotonic module (remember that its weights might sum to more than 1 due to the previous treatments). In this case, we extract most of the benefits of the previous training. Nonetheless this is the most hazardous and unstable version; indeed, there might be some extreme behaviours caused by the singularity at the threshold. These involves, notably, exploding precision parameters (η).

Note that, in any case, it is unsuitable to stop training right after re-characterization, even for the third procedure. Indeed, as the process is quite violent and non-continuous (replacing a whole subnetwork), it is highly unlikely that the model thus obtained be optimal a priori. More epochs are thus then needed on the new network in order to make the model fit the data better with its new structure.

6.4.7 Single-Valleyed Marginal Utility

A module $\tilde{\mathrm{Q}}$ is a valid single-valleyed marginal utility module if and only if there exists a single-peaked marginal utility module $\tilde{\mathrm{Q}}'$ such that:

$$\forall x \in \mathbb{R},\ \tilde{\mathrm{Q}}(x) = 1 - \tilde{\mathrm{Q}}'(x)$$

As a consequence, just as we did for the non-increasing marginal utility module, we implement a single-valleyed marginal utility module by returning $1 - \tilde{\mathrm{Q}}'$, with $\tilde{\mathrm{Q}}'$ a single-peaked marginal utility module. Note that, again, all gradients must be multiplied by -1 during backpropagation.

The recharacterization procedures are, of course, adapted as well by exploiting the symmetries.

6.4.8 Ensuring the Validity of the Parameterization

We present here the representation theorem for the bitonic marginal utility modules.

Theorem 5. *Let $\mathcal{E}^*_{\text{SP}}$ be the set of functions representable by a non-recharacterized single-peaked module defined, as above, by two monotonic submodules and a threshold, and whose parameters satisfy* (4.1), (4.2), (4.3) *(for both submodules), and* (6.4). *Let $\mathcal{E}_{\text{SP}}$ the set of all valid single-peaked marginal utilities.*

Then $\mathcal{E}^*_{\text{SP}}$ is a dense part of $\mathcal{E}_{\text{SP}}$.

Proof : 1. Proof that $\mathcal{E}^*_{\text{SP}} \subseteq \mathcal{E}_{\text{SP}}$

This amounts to showing that any valid $\tilde{\text{Q}} \in \mathcal{E}^*_{\text{SP}}$ is a valid single-peaked marginal utility. We know that $\tilde{\text{Q}}$ is composed of two monotonic submodules Q and Q′ and a threshold $T \in \mathbb{R}$.

We know from Theorem 4 and Corollary 2 that the monotonic submodules represent valid monotonic marginal utilities of their respective types. Given the definition of bitonic marginal utilities, we have that $\tilde{\text{Q}}$ is a valid single-peaked marginal utility.

2. Proof that $\mathcal{E}^*_{\text{SP}}$ is dense in $\mathcal{E}_{\text{SP}}$

Let $f \in \mathcal{E}_{\text{SP}}$ be defined by the triple $\langle m, m', T\rangle$ that satisfies $(m, m') \in (\mathcal{E}_{\text{ND}} \times \mathcal{E}_{\text{NI}})$ and $T \in \mathbb{R}$ the threshold.

Then we know by Theorem 4 and Corollary 2 that m and m' can be approximated by monotonic modules $\text{Q} \in \mathcal{E}^*_{\text{ND}}$ and $\text{Q}' \in \mathcal{E}^*_{\text{NI}}$ with arbitrary precision. Building a single-peaked module $\tilde{\text{Q}}$ with T as a threshold and Q, Q' as its submodules, we can approximate f to any precision.

This concludes the proof. □

Corollary 3. *$\mathcal{E}^*_{\text{SV}}$ is a dense part of $\mathcal{E}_{\text{SV}}$.*

Theorem 6. *The set of functions that a single-peaked module $\tilde{\text{Q}}$ can represent after training is dense in $\mathcal{E}_{\text{ND}} \cup \mathcal{E}_{\text{NI}} \cup \mathcal{E}_{\text{SP}}$.*

Proof : There are two possibilities. Either $\tilde{\text{Q}}$ remained a single-peaked marginal utility module (no recharacterization). Then that the set of functions that $\tilde{\text{Q}}$ can take is $\mathcal{E}^*_{\text{SP}}$.

Otherwise, $\tilde{\text{Q}}$ is now either a non-decreasing or a non-increasing marginal utility. The set of functions that $\tilde{\text{Q}}$ can now represent is now $\mathcal{E}^*_{\text{ND}} \cup \mathcal{E}^*_{\text{NI}}$.

Thus, the total set representable is $\mathcal{E}^*_{\text{ND}} \cup \mathcal{E}^*_{\text{NI}} \cup \mathcal{E}^*_{\text{SP}}$.

Theorems 5 and 4 and Corollary 2 yield that this set is a dense part of $\mathcal{E}_{\text{ND}} \cup \mathcal{E}_{\text{NI}} \cup \mathcal{E}_{\text{SP}}$, concluding the proof. □

Corollary 4. *The set of functions that a single-valley module* $\tilde{\text{Q}}$ *can represent after training is dense in* $\mathcal{E}_{\text{ND}} \cup \mathcal{E}_{\text{NI}} \cup \mathcal{E}_{\text{SV}}$.

6.5 Marginal Utility Selector Module

The use of the monotonic or bitonic marginal utility modules presented in the previous sections supposes that the type be given as a parameter of the model (in particular as a single-valleyed model cannot be re-characterized as a single-peaked and vice-cersa). Nonetheless, there might be cases where we wish the model to select by itself the most adapted type of marginal utility during training. We implemented modules which fulfill this very task. We call them *marginal utility selectors*, or selectors, for short.

6.5.1 Implementation as a Neural Module

A selector $\tilde{\text{Q}}$ is a multilayer perceptron composed of two utility modules, Q and Q′, along with a switch $t \in \mathbb{R}$. Q and Q′ must have different types, among ND, NI, SP and SV. Then, the output of the module given an input $x \in \mathbb{R}$ is computed as:

$$\tilde{\text{Q}}(x) = \sigma(t)\text{Q}(x) + (1 - \sigma(t))\text{Q}'(x)$$

Where σ is the logistic sigmoid defined before, ensuring $\sigma(t) \in [0, 1]$. We see here that $\sigma(t)$ plays the role of a switch, favoring Q when $\sigma(t) \sim 1$ and Q′ when $\sigma(t) \sim 0$. It is common for a sigmoid to play such a role, given its behaviour, saturating in 0 on one side of $\mathbb{R}$, and to 1 on the other, while remaining smoothly differentiable.

Then, both Q and Q′, and t will be learned. The training of t, and the direction whereto it tends, will determine which of Q and Q′ will be selected by the model.

6.5.2 Backpropagation

At each backpropagation step, the parameters of Q and Q′ are updated normally (that is, as described above for their respective types), with the nuance that the propagated gradients are weighed by $\sigma(t)$ for Q and $(1-\sigma(t))$ for Q′. The gradient of t is thus computed as:

$$\begin{aligned}\frac{\partial \text{err}}{\partial t} &= \frac{\partial \text{err}}{\partial \tilde{\text{Q}}(x)} \cdot \frac{\partial \tilde{\text{Q}}(x)}{\partial \sigma(t)} \cdot \frac{\partial \sigma(t)}{\partial t} \\ &= \frac{\partial \text{err}}{\partial \tilde{\text{Q}}(x)} \cdot (\text{Q}(x) - \text{Q}'(x))\, \sigma(t)(1-\sigma(t))\end{aligned}$$

6.5.3 Re-characterization

It is obvious that, as long as $\sigma(t)$ is neither 0 or 1, $\tilde{\text{Q}}$ has no reason to be a valid marginal utility, as it is simply a mixture of valid ones. As a consequence, it is necessary to establish a criterion to select either marginal utility. The main difference with the re-characterization of single-peaked utilities is that, here, selecting either Q or Q′ is necessary in order to have a valid model. As a consequence, the selection must be forced at some point during the training, for instance, when a certain number of epochs is reached.

We apply the following strategy: we choose a margin parameter ϵ and a stability parameter k. Then, if $\sigma(t)$ remains ϵ-close to 0 (resp 1) for k epochs, Q′ (resp Q) is chosen, and takes the place of u_i for the subsequent iterations. This is very close to the threshold-based re-characterization criterion from Section 6.4.6. The methods for replacing the $\tilde{\text{Q}}$ with the selected sub-module can be either of those evoked in 6.4.6.3.

In any case, it is preferable to keep training for some epochs after such a re-characterization (or to re-train completely using the new marginal utility), as there are chances that the selected module will not be in its optimal state at that point.

6.5.4 Representable Functions

We write $\text{Sel}(t_1, t_2)$ the type of a Selector module with submodules of types t_1 and t_2. We introduce the function type to yield the type of a marginal utility (among $\text{ND}, \text{NI}, \text{SP}, \text{SV}$). We say that a type is *reachable* by module Q if Q can become a module of this type, through re-characterization. Table 6.1 shows which types are reachable for each type of modules. We write $\text{R}(M)$ the set of types reachable by a module Q.

Theorem 7. *The set of functions representable by a module* $\tilde{\text{Q}} = \langle M, \text{Q}' \rangle$ *of type* $\text{Sel}(t_1, t_2)$ *after training is a dense part of* $\bigcup_{t \in \text{R}(M) \cup \text{R}(\text{Q}')} \mathcal{E}_t$.

Proof : After training, $\tilde{\text{Q}}$ has selected one of its submodule. Without loss of generality, assume $\sigma(t)$ 1 and Q was selected. Thus, either Q was not recharacterized during training (and is thus still of type t_1), or it was (and is thus of either type in $\text{R}(M)$).

type(M)	R(M)
ND	ND
NI	NI
SP	SP, ND, NI
SV	SV, ND, NI
Sel(SP, SV)	SP, SV, ND, NI
Sel(ND, NI)	ND, NI

Table 6.1: Types reachable by each type of marginal utility module, through recharacterization

Considering that type is in $\{\text{ND}, \text{NI}, \text{SV}, \text{SP}\}$, we have, by Theorem 4, 5 or 6 or their corrolaries, that the set of functions thus representable is a dense part of $\bigcup_{t \in \text{R}(M)} \mathcal{E}_t$. Assuming Q′ had been selected, we would have a dense part of $\bigcup_{t \in \text{R}(M) \cup \text{R}(\text{Q}')} \mathcal{E}_t$ instead.

Hence the conclusion. □

6.5.5 Discussion

In practice, one can use any two marginal utility modules for a selector Q and Q′. The two most usual cases, nonetheless, would be

- (1): Q non-decreasing and Q′ non-increasing, when u_i is known to be monotonic
- (2): Q single-peaked and Q′ single-valleyed

Note that case (2) is the most general: this module learn any of the four types evoked, through recharacterization (see Table 6.1 and Theorem 7).

CHAPTER 7

AGGREGATOR MODULES

Contents

7.1 Role of an Aggregator Module

We have seen in the previous chapter how to implement marginal utility functions as neural networks. That is, given one utiliy module u_i for each $i \in N$ and an alternative $\mathbf{x} \in X$, we obtain the vector $\mathbf{u} := \mathcal{U}(\mathbf{x}) := (u_1(x_1), ..., u_n(x_n))$ by applying each marginal utility on the associated attribute. $\mathbf{u}$ is the vector which represents how satisfying $\mathbf{x}$ is on each criterion.

Note that, later in this chapter, we abuse the notation u_i to mean either the ith marginal utility function or the ith component of $\mathbf{u}$.

The following step, as described in Section 2.4.2, is to *aggregate* those marginal satisfactions into a final, global score for the whole alternative.

We have presented several aggregators, based on the Choquet integral (CI). We will show here how they are implemented as neural modules. As for the marginal utility modules, we want the set of functions represented by an aggregator module to be dense in the set of aggregators of the same type. We recall that we only work with normalized fuzzy measures here.

We work on three types of aggregators, all variants of the Choquet integral. These types are:

- the general (or non-additive) Choquet integral
- the 2-additive Choquet integral
- the $0-1$-based 3-additive Choquet integral

We call these types, respectively, $\mathrm{CG}, \mathrm{C2}, \mathrm{C3}$.

7.2 General Choquet Integral

The first module we implement is that of the general Choquet integral.

7.2.1 Reminder and Validity Constraints

A CI parameterized by a fuzzy measure μ can be written as:

$$C_\mu(\mathbf{u}) = \sum_{i=1}^{n} \left(u_{\tau(i)} - u_{\tau(i-1)}\right) \mu(A_i)$$

with the u_i permuted such that $0 =: u_{(1)} \leq u_{(2)} \leq ... \leq u_{(n)}$, and $A_i = \{(i), (i+1), ..., (n)\}$.

Ensuring the validity of such a model is equivalent to ensure the validity of the FM μ, which come down to:

- **Monot:** $\forall A, B \subseteq N,\ B \subseteq A \Rightarrow \mu(B) \leq \mu(A)$
- **Normal1:** $\mu(\emptyset) = 0$
- **Normal2:** $\mu(N) = 1$

7.2.2 Representating the General FM

Lemma 10. *Given the normalization properties **Normal1** and **Normal2**, the monotonicity property is equivalent to:*

$\forall A \subseteq N, \forall B \subseteq A$ *such that* $|B| = |A| - 1$, *we have* $\mu(B) \geq \mu(A)$ *(*).*

Proof : It is obvious that **Monot** $\Rightarrow$ (*), as (*) is a sub-case of **Monot**.

The other direction of the equivalence is proven by immediate induction: given $B \subseteq A$, there exists a path $B = B_0, B_1, ..., B_{|A|-|B|} = A$ such that $B_k \subseteq B_{k+1}$ and $|B_k| = |B| + k$. Thus we have, by (*):

$\forall k \in \{1, |A| - |B|\},\ \mu(B_k) \geq \mu(B_{k-1})$, giving **Monot** by transitivity.□

Theorem 8. *Let $\mathcal{M}_{\text{CG}}$ the set of all fuzzy measures on N.*

Let F the set of all set functions on 2^N with values in the unit interval, such that $f(\emptyset) = 0$ and $f(N) = 1$:

$$F := \{f,\ f : 2^N \to [0,1], f(\emptyset) = 0, f(N) = 1\}$$

Now, given $f \in F$, we write ν_f the set function computed for all $\emptyset \neq A \subseteq N$:

$$\nu_f(A) = f(A) \left(1 - \max_{i \in A} \left(\nu_f \left(A \setminus \{i\}\right)\right)\right) + \max_{i \in A} \left(\nu_f \left(A \setminus \{i\}\right)\right)$$

and $\nu_f(\emptyset) = 0$.

We define by $\mathcal{M}^*_{\text{CG}} := \{\nu_f,\ f \in F\}$. Then $\mathcal{M}^*_{\text{CG}} = \mathcal{M}_{\text{CG}}$

Proof : 1. $\mathcal{M}^*_{\mathrm{CG}} \subseteq \mathcal{M}_{\mathrm{CG}}$

This amounts to show that, for all $\nu_f \in \mathcal{M}^*_{\mathrm{CG}}$, ν_f satisfies **Normal1**, **Normal2** and **Monot**.

Normal1 and **Normal2**: let $\nu_f \in \mathcal{M}^*_{\mathrm{CG}}$. By definition of ν_f, we have **Normal1**. By definition of f, we have $f(N) = 1$; thus:

$$\begin{aligned}\nu_f(N) &= f(N)\left(1 - \max_{i\in N}\left(\nu_f\left(N \setminus \{i\}\right)\right)\right) + \max_{i\in N}\left(\nu_f\left(A \setminus \{i\}\right)\right)\\ &= 1 - \max_{i\in N}\left(\nu_f\left(N \setminus \{i\}\right)\right) + \max_{i\in N}\left(\nu_f\left(N \setminus \{i\}\right)\right)\\ &= 1\end{aligned}$$

which gives us **Normal2**.

Monot: let $\nu_f \in \mathcal{M}^*_{\mathrm{CG}}$. We first need to show that, $\forall A \subseteq N$, $0 \leq \nu_f(A) \leq 1$. We do this by strong induction on the cardinality of A; let our induction hypothesis H_t be: $\forall A \subseteq N,\ |A| \leq t$, we have $0 \leq \nu_f(A) \leq 1$. H_0 is true as $\nu_f(\emptyset) = 0$. Then, assuming H_t, let A such that $|A| = t + 1$:

$$\begin{aligned}&\nu_f(A) = \max_{i\in A}\left(\nu_f\left(A \setminus \{i\}\right)\right)\left(1 - f(A)\right) + f(A)\\ &\Rightarrow 0(1 - f(A)) + f(A) \leq \nu_f(A) \leq (1 - f(A)) + f(A) \text{ by } H_t \text{ and } (1 - f(A)) \geq 0\\ &\Rightarrow f(A) \leq \nu_f(A) \leq 1\\ &\Rightarrow 0 \leq \nu_f(A) \leq 1 \text{ by } f(A) \geq 0\end{aligned}$$

Now, we can prove **Monot**; let $\emptyset \neq A \subseteq N$. Then, we have:

$$f(A)\left(1 - \max_{i\in A}\left(\nu_f\left(A \setminus \{i\}\right)\right)\right) \geq 0$$

by the previous result and the fact that $f(A) \geq 0$, thus:

$$\nu_f(A) \geq \max_{i\in A}\left(\nu_f\left(A \setminus \{i\}\right)\right)$$

By Lemma 10, and as we have shown **Normal1** and **Normal2**, we have **Monot**, which conludes this part of the proof. We have thus shown that any function in $\mathcal{M}^*_{\mathrm{CG}}$ is a valid FM

2. $\mathcal{M}_{\mathrm{CG}} \subseteq \mathcal{M}^*_{\mathrm{CG}}$

Let μ a fuzzy measure on N. Let $f \in \mathcal{M}_{\mathrm{CG}}$ such that, $\forall \emptyset \neq A \subseteq N$:

$$f(A) = \mu(A) - \max_{i\in A}\left(\mu\left(A \setminus \{i\}\right)\right)$$

Then $\mu = \nu_f$; thus $\mu \in \mathcal{M}^*_{\mathrm{CG}}$, which concludes the proof. □

The advantage is that this new representation allows to have very simple representation of the constraints: ensuring that all values $f(A)$ remain in the unit interval is enough to represent any, and only, fuzzy measures. We build a neural module based on this representation.

7.2.3 Representation of the CI as a Neural Module

We present here the hierarchy which allows us to exploit the representation presented in Theorem 8 in order to implement a neural module. Note that this module exploits more than the simple neurons presented in Chapter 3. Nonetheless, it retains the general principles, along with the ability to behave as a backpropagating cell presented in [Lecun et al., 1998].

7.2.3.1 Structure and Parameters

A General Choquet Module Q on N possesses the following attributes:

- an input layer inp with n neurons $(\mathrm{inp}_1, ..., \mathrm{inp}_n)$
- a first hidden layer sor with n neurons $(\mathrm{sor}_1, ..., \mathrm{sor}_n)$
- a second hidden layer dif with n neurons $(\mathrm{dif}_1, ..., \mathrm{dif}_n)$
- an output layer out with 1 neuron
- a set of parameters $P \in \{p_A \in \mathbb{R},\ A \subseteq N\}$, with $p_\emptyset = -\infty$ and $p_N = \infty$

The idea behind the set of parameters p_A is that we wish to have a set of parameters $\{a_A \in \mathbb{R},\ A \subseteq N\}$, which would serve the role of the $f(A)$ in Theorem 8. These a_A need to be kept in the unit interval, which might require clipping or other violent procedures. Such procedures make the training unstable. As a consequence, the approach we have is to allow the learned variables p_A to cover the entire real set unhindered, and then to compute the a_A as a smooth, constrained function of p_A. In practice, we use a logistic sigmoid, such that $\forall A \subseteq N,\ a_A = \sigma(p_A)$. The sigmoid has the nice properties of being smooth, easily and quickly differentiable, and monotonic, which make this a suitable intermediate step for stabilizing training. Moreover, it allows to bound the value by design, rather than using a specific step for renormalization.

As we have the a_A easily computed, we also assume that we have the corresponding μ at hand, computed as presented in the theorem.

7.2.3.2 Forward Propagation

Assume a vector $\mathbf{u} = (u_1, ..., u_n)$ is given as an input. The forward propagation happens as follows:

1. the input inp is set to x, i.e. $\text{inp}_i = u_i \ \forall i \in N$
2. a sorting operation is performed on inp by a *sorter*, returning the ordered values along with the permutation $\tau_{\mathbf{u}} : N \to N$ s.t. $\forall i \in N,\ \text{inp}_{\tau_{\mathbf{u}}(1)} \leq \text{inp}_{\tau_{\mathbf{u}}(2)} \leq ..., \leq \text{inp}_{\tau_{\mathbf{u}}(n)}$. The sorter also propagates the $\tau_{\mathbf{u}}$ value to the output node, and retains memory of τ in order to backpropagate efficiently
3. sor receives the output values of the sorter, such that $\forall i \in N,\ \text{sor}_i = \text{inp}_{\tau_{\mathbf{u}}(i)}$,
4. dif is not fully-connected to sor; in fact, it acts as a 1d convolution with kernel $(1, -1)$ s.t. $\text{dif}_1 = \text{sor}_1$ and for all $i > 1$ $\text{dif}_i = \text{sor}_i - \text{sor}_{i-1}$
5. this step is the trickiest. Indeed, we need to do some *weights selection* based on the $\tau_{\mathbf{u}}$ sent by the sorter. To do this, we retrieve the vector $w^{\mathbf{u}}$ of elements from the pool μ, defined as

$$w^{\mathbf{u}} := (\mu(\{\tau_{\mathbf{u}}(2), ..., \tau_{\mathbf{u}}(n)\}), \mu(\{\tau_{\mathbf{u}}(3), ..., \tau_{\mathbf{u}}(n)\}), ..., \mu(\{\tau_{\mathbf{u}}(n)\}))$$

   . We also need to keep in memory which parameters were used, so that we can backpropagate later
6. the passage from dif to out is a simple dot product $\text{out} \leftarrow w^{\mathbf{u}} \cdot \text{dif}$

The main idea behind the last layer is that the weights are conditioned by the input. That is, the fuzzy measure is, in itself, a *pool* of weights, which will only be used for some inputs and not others. This is equivalent to choosing a linear region of the CI based on the input. This process is illustrated in Figure 7.1.

7.2.4 Validity of the Module

We write and prove the following representation theorem:

Theorem 9. *Let $\mathcal{E}_{\text{CG}}$ the set of all general CIs on N. We define by $\mathcal{E}^*_{\text{CG}}$ the set of functions that a general Choquet integral module* Q *can represent. Then $\mathcal{M}^*_{\text{CG}}$ is dense in $\mathcal{M}_{\text{CG}}$*

Proof : $1.\mathcal{E}^*_{\text{CG}} \subseteq \mathcal{E}_{\text{CG}}$:

Assume a general CI module Q, with P as its pool of parameters. Let $P' := \{a_A := \sigma(p_A),\ A \subseteq N\}$. Then, we have, $\forall A \subseteq N,\ 0 \leq a_A \leq 1$, along with $a_\emptyset = 0$

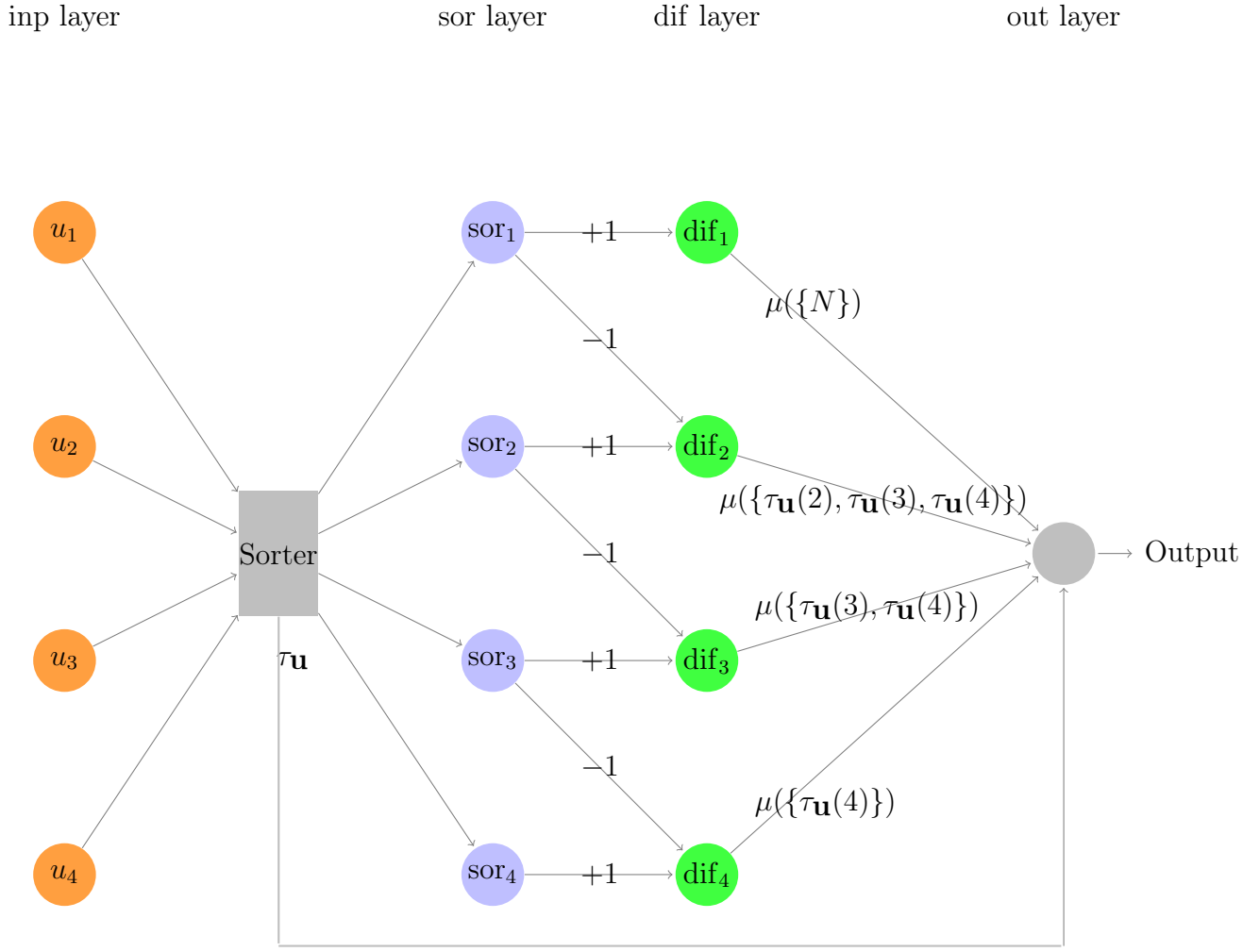


Figure 7.1: A general Choquet integral module with 4 inputs

and $a_N = 1$. By Theorem 8, we know that P' represents a valid general fuzzy measure μ, which we can thus compute. Then, for a given $\mathbf{u} \in U$, we follow the same steps as presented in Section 7.2.3.2, and show that the result is, as expected $C_\mu(\mathbf{u})$):

1. $\text{inp} = \mathbf{u}$
2. $\tau_\mathbf{u}$ is computed, such that $u_{\tau_\mathbf{u}(1)} \leq ... \leq u_{\tau_\mathbf{u}(n)}$
3. $\text{sor} = (u_{\tau_\mathbf{u}(1)}, ..., u_{\tau_\mathbf{u}(n)})$
4. $\text{dif} = (u_{\tau_\mathbf{u}(1)}, u_{\tau_\mathbf{u}(2)} - u_{\tau_\mathbf{u}(1)}, ..., u_{\tau_\mathbf{u}(n)} - u_{\tau_\mathbf{u}(n-1)})$
5. $w^\mathbf{u}$ is computed as:

$$w^\mathbf{u} := (\mu(\{\tau_\mathbf{u}(2), ..., \tau_\mathbf{u}(n)\}), \mu(\{\tau_\mathbf{u}(3), ..., \tau_\mathbf{u}(n)\}), ..., \mu(\{\tau_\mathbf{u}(n)\}))$$

6. out is computed as:

$$w^\mathbf{u} \cdot \text{dif} = \sum_{i=1}^{n} w_{u_i} \text{dif}_i = u_{\tau_\mathbf{u}(1)} + \sum_{i=2}^{n} (u_{\tau_\mathbf{u}(i)} - u_{\tau_\mathbf{u}(i-1)}) \mu(\{\tau_\mathbf{u}(i), ..., \tau_\mathbf{u}(n)\})$$

We thus see that the output is exactly the formula given in Equation (2.10): thus, Q does represent a valid parameterized by a valid FM μ.

2.$\mathcal{E}^*_{\text{CG}}$ is dense in $\mathcal{E}_{\text{CG}}$:
Let C_μ a CI parameterized by μ. If we can approximate μ arbitrarily closely, then we can represent C_μ as accurately as desired. This is equivalent, by Theorem 8 to being able to approximate any $P' := \{a_A \in [0, 1]\}$ as closely as possible. This is immediate, as the output domain of the logistic sigmoid is the open unit interval; thus, $\forall a \in [0, 1]\,,\ \exists p \in \mathbb{R}$ s.t. $\sigma(p)$ is arbitrarily close to a. Thus, our architecture can represent any general Choquet integral to any desired precision.

The proof is thus concluded. □

7.2.5 Backpropagation

We have seen how to implement this module; now we explain how we train it. We write $\text{Q}(\mathbf{u})$ the output of a general CI module Q given the input $\mathbf{u}$. We assume that we are given $\frac{\partial \text{err}}{\partial M(\mathbf{u})}$ the gradient of the error, which was backpropagated back to the module.

7.2.5.1 Gradient w.r.t. the p_A

Assume a set $A \subseteq N$ s.t. $\mu(A)$ is part of $w^{\mathbf{u}}$. This means that:

$$\exists i \in N \text{ s.t. } A = \{\tau_{\mathbf{u}}(i), ...\tau_{\mathbf{u}}(n)\}$$

We recall that $p_A \in \mathbb{R}$ is the parameter such that $a_A = \sigma(p_A)$, as defined in Section 7.2.3.1. Then:

$$\begin{aligned}\frac{\partial \text{err}}{\partial p_A} &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \frac{\partial M(\mathbf{u})}{\partial \mu(A)} \cdot \frac{\partial \mu(A)}{\partial a_A} \cdot \frac{\partial a_A}{\partial p_A} \\ &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \text{dif}_i \cdot 1 \cdot a_A(1 - a_A)\end{aligned}$$

Nonetheless, it is to be noted that $\mu(A)$, in this expression, does not depend only on a_A. Indeed, we recall:

$$\mu(A) = a_A + \max_{j \in A} \left(\mu\left(A \setminus \{j\}\right)\right)$$

Writing $\emptyset = B_0, ..., B_{|A|} = A$, a sequence of sets such that for each j, $|B_j| = j$, and $B_j = \underset{B \subseteq B_{j+1}}{\arg\max}(\mu(B))$, then we have:

$$\mu(A) = \sum_{j=0}^{|A|} a_{B_j}$$

It is thus logical to apply a similar gradient to all of the p_B with $B \in B_0, ..., B_{|A|}$.

7.2.5.2 Gradient w.r.t. the u_i

Now that we can update the parameters, it is important to be able to backpropagate to the input, so that modules that precede Q can be updated themselves. For the sake of commodity, we can add a ghost term $w^{\mathbf{u}}_{n+1} = 0$ so that we avoid an exception for the extremum.

$$\begin{aligned}\frac{\partial \text{err}}{\partial u_i} &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \frac{\partial M(\mathbf{u})}{\partial \text{dif}} \cdot \frac{\partial \text{dif}}{\partial \text{sor}} \cdot \frac{\partial \text{sor}}{\partial u_i} \\ &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \left(w^{\mathbf{u}}_{\tau(i)} - w^{\mathbf{u}}_{\tau(i)+1}\right)\end{aligned}$$

7.2.5.3 Normalization After Parameters Update

In order for the model to work, we see that we need the measure μ to be recomputed after each parameter update. Obviously, this is a costly operation, as there is an exponential number of variables to compute. We might thus be tempted to avoid recomputing this number of parameters too often, for instance by training only with large batches.

Nonetheless, there is no other normalization step required in this case.

7.2.6 Discussion

This module is very interesting in the sense that it allows to represent any (and only) Choquet integral with dimension n. As a consequence, it is a highly expressive model, which nonetheless satisfies the monotonicity and normalization constraints suitable for such a decision model.

On the other hand, it has an obvious flaw in its high number of parameters. Indeed, as we saw, there needs to be 1 parameter for any subset of N (with the exception of N itself and $\emptyset$ whose measures are fixed by the normalization constraints). This brings the total amount to $2^n - 2$. There are three main issues raised by this exponential number of parameters.

The clear first issue is the computational cost, which quickly becomes intractable as the number of criteria increases. While this issue might be contained on some real problems which involve a small number of criteria (10 or less), this type of aggregator becomes unsuitable for large-dimensional inputs.

The second issue is the effect that such a large number of parameters has on interpretability. In order to verify or explore the model, an expert would have to look at all parameters, as they hold the information about the relative importance and interactions among criteria. It is obviously time-consuming for a person to look at hundreds of parameters (or thousands, if $n \geq 10$). Even given all of that time, it might still be very hard for a person to grasp the meaning between each parameter, and thus to interpret the model. Indeed, while interactions between pairs of variables is easily understandable, interaction parameters among the members of a coalition of 3 or more elements (on top of the pairwise interactions) might be hardly exploitable for getting an interpretation out of. In particular, as there is a parameter for any coalition.

Finally, from a purely machine-learning point of view, the high number of parameters is also an issue. Indeed, unlike large-scale problems with millions of user data, or large amounts of labeled examples, MCDA problems tend to generate small amounts of data. This is particularly true in safety-critical systems, as the data is often collected from past choices by an expert from a given field. Due to the limited number of expert (data sources) and the limited amount of data that

can be processed by a single human, the size of the available datasets is often small. This, combined with a large number of parameters, could lead to heavy overfitting by the model. In particular, there might be many possible models that can represent a small dataset. This, in turn, leads to having no trust in the model, as a very different one could perform just as well.

We explore below more restricted class of the Choquet Integral, in order to tackle both of these issues.

7.3 2-additive Choquet Integral

7.3.1 General considerations

We already presented the 2-additive Choquet integral in Section 2.5.4. It is simply a Choquet integral parameterized by a 2-additive fuzzy measure. That is, given the Möbius m values of a 2-additive FM μ:

$$\forall S \subseteq N, |S| > 2 \Rightarrow m(S) = 0$$

This means that such a FM can represent interactions of up to two criteria at a time. Moreover, as we will see, it reduces the number of degrees of freedom of the model from exponential to a quadratic function of n.

We recall that we write $\wedge$ the min operator (i.e. $a \wedge b = \min(a, b)$, and $\vee$ the max.

7.3.2 Möbius-values-based parameterization

From equation (2.11) a 2-additive CI can be written as:

$$\begin{aligned} C_\mu(\mathbf{u}) &= \sum_{S \subseteq N} m(S) \bigwedge_{i \in S} (u_i) \\ &= \sum_{\substack{S \subseteq N \\ |S|=1}} m(S) \bigwedge_{i \in S} (u_i) + \sum_{\substack{S \subseteq N \\ |S|=2}} m(S) \bigwedge_{i \in S} (u_i) \\ &= \sum_{i=1}^{n} m(\{i\}) u_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} m(\{i,j\})(u_i \wedge u_j) \end{aligned} \tag{7.1}$$

For simplicity of writing, we write $m_i = m(\{i\})$ and $m_{i,j} = m(\{i,j\})$. The parameterization in equation (7.1) is very interesting from an interpretability point of view. Indeed, it reduces the number of parameters (the Möbius values) to $\frac{n(n+1)}{2}$ (n singletons, $\frac{n(n-1)}{2}$ pairs of criteria). This is exactly the number of independent

degrees of freedom at the model, and is thus the minimal number of parameters to ensure a full covering of the space of 2-additive Choquet integrals.

Nonetheless, there are some constraints to ensure the validity of the model. Just like in the previous section, these constraints are **Monot**, **Normal1**, and **Normal2**. While the normalization constraints are easy to impose on the model (namely, setting $m_\emptyset = 0$ and making sure all the parameters sum to one),the monotonicity constraint is problematic. Re-writing **Monot** in the 2-additive settings give us, $\forall A, B \subseteq N$, such that $B \subseteq A$.

$$\begin{aligned}
\mu(B) \leq \mu(A) &\iff \forall A, B \subseteq N,\ B \subseteq A \\
&\iff \sum_{S \subseteq B} m(S) \leq \sum_{S \subseteq A} m(S) \\
&\iff \sum_{\substack{S \subseteq B \\ |S| \leq 2}} m(S) \leq \sum_{\substack{S \subseteq A \\ |S| \leq 2}} m(S) \\
&\iff \sum_{\substack{S \subseteq A, S \not\subseteq B \\ |S| \leq 2}} m(S) \geq 0
\end{aligned} \tag{7.2}$$

As we can see, each constraint thus obtained is ensuring a certain sum is non-negative. Nonetheless, there is one such constraint for each pair $B \subseteq A \subseteq N$. By Lemma 10, we can restrict this only to pairs such that $|B| = |A| - 1$; nonetheless, that the number of constraints to verify remains exponential, at $n2^n - 1$.

This brings along all of the computational issues that we have seen for the general CI, and limits the advantage of using 2-additive CIs. This motivates the use of another representation of the 2-additive CI.

7.3.3 Weights-based parameterization

As a consequence, we opt for another parameterization. According to [Grabisch, 1997b], a 2-additive Choquet integral on $\mathbf{u} = (u_1 \dots u_n)$ can be written as follows:

$$C_\mu(\mathbf{u}) = \sum_{i=1}^{n} w_i u_i + \sum_{1 \leq i < j \leq n} \left(w^{\wedge}_{i,j}(u_i \wedge u_j) + w^{\vee}_{i,j}(u_i \vee u_j) \right) \tag{7.3}$$

where $\wedge$ and $\vee$ still denote the min and max operators, respectively. The fuzzy measure μ is represented by the weights parameters of the form w_i, $w^{\vee}_{i,j}$ and $w^{\wedge}_{i,j}$, such that:

- w_i for each $i \in N$
- $w^{\wedge}_{i,j}$ for each pair $i, j \subseteq N$ with $i < j$

- $w^{\vee}_{i,j}$ for each pair $i, j \subseteq N$ with $i < j$

There are thus n weights of the first type, $\frac{n(n-1)}{2}$ of the second type, and $\frac{n(n-1)}{2}$ of the third type. In total, this means that we have n^2 parameters (roughly twice as many as we have degrees of freedom).

Note that these weights are yet another representation of a 2-additive fuzzy measure μ. Given the weights, the Möbius values of this same FM can be computed as:

$$\begin{aligned} \forall i \in N,\ m_i &= w_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} w^{\vee}_{i,j} \\ \forall i, j \in N,\ i \neq j,\ m_{i,j} &= w^{\wedge}_{i,j} - w^{\vee}_{i,j} \end{aligned} \tag{7.4}$$

which, through Equation (2.6), gives, for $S \subseteq N$:

$$\mu(S) = \sum_{i \in S} \left(w_i + \sum_{\substack{j \in N \\ j \neq i}} w^{\vee}_{i,j} \right) + \sum_{i \in S} \sum_{\substack{j \in S \\ j > i}} \left(w^{\wedge}_{i,j} - w^{\vee}_{i,j} \right) \tag{7.5}$$

In order to ensure validity, we need to impose the following constraints on the weights:

$$\begin{aligned} \forall i \in N,\ w_i &\geq 0 \\ \forall i, j \in N,\ i \neq j,\ w^{\wedge}_{i,j} \geq 0 \text{ and } \ w^{\vee}_{i,j} &\geq 0 \end{aligned} \tag{7.6}$$

$$\sum_{i=1}^{n} w_i + \sum_{1 \leq i < j \leq n} \left(w^{\wedge}_{i,j} + w^{\vee}_{i,j} \right) = 1 \tag{7.7}$$

Normal1 is yielded by design. **Normal2** is given by Equation (7.7). By equation (7.6), we have **Monot**.

We show in this section that, given the set of weight W, the expression in (7.3) is exactly the same function as that given in (7.1), which is a known result [Grabisch, 2016].

Let $\mathbf{u} \in [0, 1]^n$:

$$
\begin{aligned}
C_\mu(\mathbf{u}) &= \sum_{i=1}^{n} m_i u_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} m_{i,j}(u_i \wedge u_j) \\
&= \sum_{i=1}^{n} \left(w_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{i,j}^{\vee} \right) u_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} (w_{i,j}^{\wedge} - w_{i,j}^{\vee})(u_i \wedge u_j) \\
&= \sum_{i=1}^{n} w_i u_i + \sum_{1 \leq i < j \leq n} w_{i,j}^{\vee} u_i + w_{i,j}^{\vee} u_j + w_{i,j}^{\wedge}(u_i \wedge u_j) - w_{i,j}^{\vee}(u_i \wedge u_j) \\
&= \sum_{i=1}^{n} w_i u_i + \sum_{1 \leq i < j \leq n} w_{i,j}^{\wedge}(u_i \wedge u_j) + w_{i,j}^{\vee}(u_i + u_j - (u_i \wedge u_j)) \\
&= \sum_{i=1}^{n} w_i u_i + \sum_{1 \leq i < j \leq n} \left(w_{i,j}^{\wedge}(u_i \wedge u_j) + w_{i,j}^{\vee}(u_i \vee u_j) \right)
\end{aligned}
$$

Thus, we have seen that the weights can parameterize a valid 2-additive CI.

Note that, while a weight vector represents a single fuzzy measure, a fuzzy measure has infinitely many representations in the space of weights. Nonetheless, for a given 2-additive fuzzy measure μ, there exists a weight representation such that, for any given pair of criteria $i, j,\ i \neq j$, either $w_{i,j}^{\wedge} = 0$ or $w_{i,j}^{\vee} = 0$. We call it the canonical weights.

It can be computed easily from the Möbius:

$$
\begin{aligned}
&\forall i, j \in N,\ i \neq j, \\
&\qquad w_{i,j}^{\wedge} = m_{i,j} \text{ if } m_{i,j} \geq 0, \text{ else } 0 \\
&\qquad w_{i,j}^{\vee} = m_{i,j} \text{ if } m_{i,j} \leq 0, \text{ else } 0 \\
&\qquad w_i = m_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{i,j}^{\vee}
\end{aligned}
\tag{7.8}
$$

The advantage of this representation, for interpretability, is that if $w_{i,j}^{\wedge} > 0$ and $w_{i,j}^{\vee} = 0$ means that $m_{i,j} > 0$, and thus that i and j work are synergetic, while the opposite means that these criteria are redundant.

Moreover, it allows to represent only, and any, 2-additive Choquet Integral, as stated in the original paper. In particular, there is a bijection between the set of valid 2-additive fuzzy measures and the set of canonical weights.

	Parameters	**Monot** Complexity	Normalization complexity
Möbius (7.3.2)	$\frac{n(n-1)}{2}$	$n2^n - 1$	2
Weights (7.3.3)	n^2	n^2	2

Table 7.1: Trade-off: Möbius-based vs weights-based parameterization

7.3.4 Implementation as a Neural Module

Now that we have seen that, given (7.6) and (7.7), representation (7.3) is suitable for representing 2-additive Choquet integral. Considering the small amount of constraints that have to be satisfied in order for the model to be valid, we use this representation for our 2-additive CI neural module.

We see that, given $\mathbf{u} \in [0,1]^n$, letting:

$$U^* = (u_1, ..., u_n, (u_1 \wedge u_2), ..., (u_1 \wedge u_2), (u_1 \vee u_2), ..., (u_{n-1} \vee u_n))$$

$$W = (w_1, ..., w_n, w^{\wedge}_{1,2}, ..., w^{\wedge}_{n-1,n}, w^{\vee}_{1,2}, ..., w^{\vee}_{n-1,n})$$

then (7.3) yields that $C_\mu(\mathbf{u}) = W \bullet U^*$ the dot product of both vectors. Our 2-additive Choquet integral module, will thus compute $C_\mu(u)$ in two steps.

1. compute U^* from the input $\mathbf{u}$
2. compute $C_\mu(\mathbf{u})$ from the hidden values U^*

Overall, a 2-additive Choquet integral over an n-dimensional utility vector $\mathbf{u} = (u_1, \dots u_n)$ is represented as a neural architecture with a single hidden layer h with n^2 neurons (one for each element of U^*), as shown in Figure 7.2. The idea is that $h(\mathbf{u}) = U^*$; thus the weights between h and the single output node have to be W.

Unlike a "traditional" layer, h is composed of three sublayers:

h^{Id} : composed of n neurons, one for each criterion $i \in N$. Each neuron, denoted Id_i, takes in a single input u_i, and has identity activation, thus returning u_i unchanged;

$h^{\wedge}$: composed of $\frac{n(n-1)}{2}$ neurons, one for each pair i, j of criteria. The neurons are denoted $h^{\wedge}_{i,j}$. They each take in two inputs u_i and u_j, and return $(u_i \wedge u_j)$. This corresponds to a min-pooling on each pair of criteria i and j;

$h^{\vee}$: composed of $\frac{n(n-1)}{2}$ neurons, one for each pair i, j of criteria. The neurons are denoted $h^{\vee}_{i,j}$. They each take in two inputs u_i and u_j, and return $(u_i \vee u_j)$. This corresponds to a max-pooling on each pair of criteria i and j;

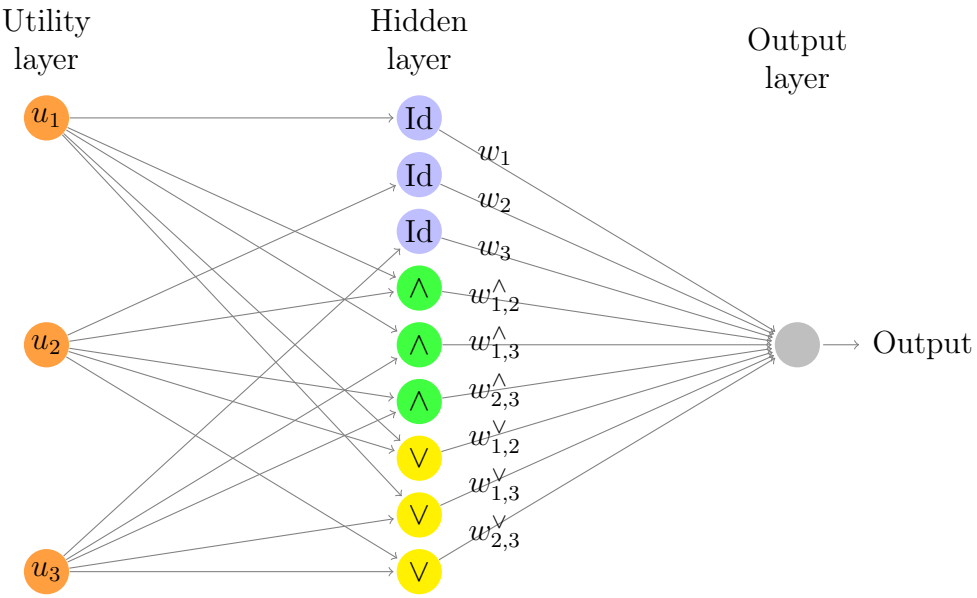


Figure 7.2: A 2-additive Choquet module with 3 inputs, involving three categories of hidden neurons noted Id, $\wedge$ and $\vee$ (see text).

Then, the output $h(\mathbf{u}) = (h_{\mathrm{Id}}(\mathbf{u}), h^{\wedge}(\mathbf{u}), h^{\vee}(\mathbf{u}))$ is the concatenation of the outputs of the three sublayers. It is then forward-propagated to the output node, weighted by W.

7.3.5 Validity of the Module

We show in this section that, given the right assumptions on the parameters of the module, it can represent any, and only, 2-additive Choquet integral.

Lemma 11. *Let* Q *be a 2-additive CI module.*

Let $W_o = (w_1, ..., w_n, w^{\wedge}_{1,2}, ..., w^{\wedge}_{n-1,n}, w^{\vee}_{1,2}, ..., w^{\vee}_{n-1,n})$ *the weights between the hidden layer and the output node.*

Given that W_o *satisfies* (7.6) *and* (7.7), Q *represents a valid 2-additive CI, parameterized by the fuzzy measure* μ *associated with* W_o.

Proof : We have seen that, by construction of h:

$$\begin{aligned} h(\mathbf{u}) &= (h_{\mathrm{Id}}(\mathbf{u}), h^{\wedge}(\mathbf{u}), h^{\vee}(\mathbf{u})) \\ &= (u_1, ..., u_n, u_n, (u_1 \wedge u_2), ..., (u_1 \wedge u_2), (u_1 \vee u_2), ..., (u_{n-1} \vee u_n)) \end{aligned}$$

thus, $h(\mathbf{u}) = U^*$.

Just like for h, we decompose W_o into three parts:

- $W_o^{\mathrm{Id}} = (w_1, ..., w_n) \in \mathbb{R}^n$
- $W_o^{\wedge} = (w_{1,2}^{\wedge}, ..., w_{n-1,n}^{\wedge}) \in \mathbb{R}^{\frac{n(n-1)}{2}}$
- $W_o^{\vee} = (w_{1,2}^{\vee}, ..., w_{n-1,n}^{\vee}) \in \mathbb{R}^{\frac{n(n-1)}{2}}$

Then, during the propagation from the hidden layer to the output layer, the dot product $W_o \bullet h(\mathbf{u})$ is computed. This yields:

$$\begin{aligned} W_o \bullet h(\mathbf{u}) &= W_o^{\mathrm{Id}} \bullet h^{\mathrm{Id}} + W_o^{\wedge} \bullet h^{\wedge} + W_o^{\vee} \bullet h^{\vee} \\ &= \sum_{i=1}^{n} w_i u_i + \sum_{1 \le i < j \le n} \left(w_{i,j}^{\wedge}(u_i \wedge u_j) + w_{i,j}^{\vee}(u_i \vee u_j) \right) \end{aligned}$$

Given that W_o satisfies (7.6) and (7.7) by hypothesis, we have exactly the definition of a valid 2-additive CI according to (7.3). The model represented by Q is thus valid. □

Lemma 12. *Let μ be a 2-additive fuzzy measure. Then, it is possible for a 2-additive CI model* Q *to be such that* $\forall \mathbf{u} \in [0,1]^n$, $\mathrm{Q}(\mathbf{u}) = C_\mu(\mathbf{u})$.

Proof : Let W any of the weight vectors associated with μ (which can be computed by using the Möbius values of μ through equations (7.1) and (7.8). We know by [Grabisch, 1997b] that W satisfies both (7.6) and (7.7).

Then, setting W as the weights for the output neuron of Q, we have, by Lemma 11 that $\forall \mathbf{u} \in [0,1]^n$, $\mathrm{Q}(\mathbf{u}) = C_\mu(\mathbf{u})$.

Thus, any 2-additive CI is representable by a certain parameterization of a 2-additive CI module. □

Theorem 10. *Letting $\mathcal{E}_{\mathrm{C2}}^*$ the domain of all functions that a 2-additive CI module can take, and $\mathcal{E}_{\mathrm{C2}}$ the domain of all 2-additive CIs. Then $\mathcal{E}_{\mathrm{C2}} = \mathcal{E}_{\mathrm{C2}}^*$.*

Proof : Lemma 11 gives us $\mathcal{E}_{\mathrm{C2}}^* \subseteq \mathcal{E}_{\mathrm{C2}}$. Lemma 12 gives us $\mathcal{E}_{\mathrm{C2}} \subseteq \mathcal{E}_{\mathrm{C2}}^*$.

Hence the equality. □

7.3.6 Ensuring the Satisfaction of the Constraints

As we have seen before, there are only a few constraints to satisfy in order for this model to be a valid 2-additive CI. Namely, these constraints are (7.7) and (7.6).

7.3.6.1 Ensuring Monotonicity

We recall that (7.6) is equivalent to **Monot** on the weights-based representation. All we need to do in this case is to make sure that all the weights in W are non-negative.

Following the method used for the marginal utility modules, one could reckon that clipping the negative weights to 0 might be the right solution. Nonetheless, we noticed that, in this setting, this violent process on a small amount of parameters causes instability during training.

In order to adress this issue, we opted for an option analog to the one we used for the general CI modules. That is, we do not directly learn the elements of W. Rather, we learn a set of n^2 latent variables $Z = (z_1, ..., z_n, z^{\wedge}_{1,2}, ..., z^{\wedge}_{n-1,n}, z^{\vee}_{1,2}, ..., z^{\vee}_{n-1,n}) \in \mathbb{R}^{n^2}$, such that, given a smoothly differentiable non-negative function $\text{pos} : \mathbb{R} \to \mathbb{R}_+$, we have:

$$\begin{aligned} \forall i \in \{1 \dots n\} : w_i &= \text{pos}(z_i); \\ \forall i, j \in \{1 \dots n\} : i < j,\ w^{\wedge}_{i,j} &= \text{pos}(z^{\wedge}_{i,j}) \\ \forall i, j \in \{1 \dots n\} : i < j,\ w^{\vee}_{i,j} &= \text{pos}(z^{\vee}_{i,j}) \end{aligned}$$

Noting that clipping negative weights to zero is the same as applying a ReLU function on each weight, we use the *softplus* function, which is a smooth version of the ReLU; it is defined as:

$$\forall x \in \mathbb{R},\ \text{pos}(x) = \ln(1 + e^x)$$

This function is illustrated in Figure 7.3.

As one can see, it quickly goes towards 0 for a negative input, and quickly goes towards the identity for a positive input. Nonetheless, the singularity of the ReLU in 0 is avoided by a smooth, differentiable transition. Moreover, the derivative of pos is easy to obtain, as:

$$\nabla\text{pos}(x) = \frac{1}{1 + e^{-x}} = \sigma(x)$$

with σ, as before, the logistic sigmoid.

Moreover, pos is bijective, and we can compute its inverse function pos^{-1}:

$$\text{pos}^{-1}(x) = \ln(e^x - 1)$$

Applying this procedure, and working with Z, we ensure that all elements of W be positive at all time, by design, and that the gradient is always well defined. This, in turn, ensures the monotonicity of the 2-additive FM represented by W.

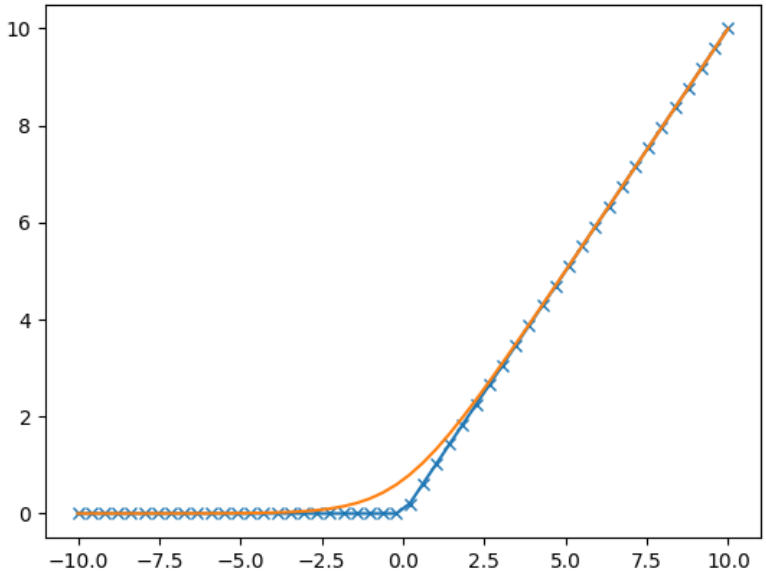


Figure 7.3: Plot of the Softplus function (single line) and ReLU (crossed line)

7.3.6.2 Ensuring Normalization

In order to ensure normalization as described in equation (7.7), we simply divide W at each epoch by the sum of its component. That is:

$$W \leftarrow \frac{W}{\sum_{i=1}^{n} w_i + \sum_{1 \leq i < j \leq n} \left(w_{i,j}^{\wedge} + w_{i,j}^{\vee} \right)}$$

Then, the Z need to be re-computed from the new values of W.

We synthetize now the steps of the process for parameters update in a 2-additive Choquet module:

1. receive the back-propagated gradient $\frac{\partial \text{err}}{\partial M(\mathbf{u})}$
2. compute the gradients for Z and $\mathbf{u}$
3. back-propagate $\frac{\partial \text{err}}{\partial \mathbf{u}}$
4. update Z through $Z \leftarrow Z - \rho_Z \frac{\partial \text{err}}{\partial Z}$, with ρ_Z the learning rate
5. update W through $W \leftarrow \text{pos}(Z)$

6. normalize W through $W \leftarrow \frac{W}{\sum_{i=1}^{n} w_i + \sum_{1 \leq i < j \leq n} \left(w^{\wedge}_{i,j} + w^{\vee}_{i,j}\right)}$

7. update Z through $Z \leftarrow \text{pos}^{-1}(W)$

At the end of the given epoch, W still represents a valid 2-additive FM, and the network is in a coherent state, as $W = \text{pos}(Z)$. We can thus proceed with a new epoch, or stop training.

7.3.7 Backpropagation

Now that we have seen how to ensure, at each step, that the module be a valid 2-additive CI, we show how to compute the gradients of each parameters, along with a synthesis of the parameters update pipeline.

7.3.7.1 Gradient w.r.t. h

Once again, we assume the gradient of the error err to have been backpropagated to the module. We write Q($\mathbf{u}$) the output of the module when the input is $\mathbf{u}$. First of all, we look at the gradients of err w.r.t. the hidden layer:

$\forall i \in N$:

$$\begin{aligned}\frac{\partial \text{err}}{\partial h_i^{\text{Id}}} &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \frac{\partial M(\mathbf{u})}{\partial h_i^{\text{Id}}} \\ &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot w_i\end{aligned}$$

$\forall i, j,\ i \neq j \in N$:

$$\begin{aligned}\frac{\partial \text{err}}{\partial h^{\wedge}_{i,j}} &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \frac{\partial M(\mathbf{u})}{\partial h^{\wedge}_{i,j}} \\ &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot w^{\wedge}_{i,j} \\ \frac{\partial \text{err}}{\partial h^{\vee}_{i,j}} &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \frac{\partial M(\mathbf{u})}{\partial h^{\vee}_{i,j}} \\ &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot w^{\vee}_{i,j}\end{aligned}$$

7.3.7.2 Gradient w.r.t. Z

We can now compute the gradients of the elements of Z:

$\forall i \in N$:

$$\begin{aligned}\frac{\partial \text{err}}{\partial z_i} &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \frac{\partial M(\mathbf{u})}{\partial w_i} \cdot \frac{\partial w_i}{\partial z_i} \\ &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot h_i^{\text{Id}} \cdot \sigma(z_i)\end{aligned}$$

$\forall i, j,\ i \neq j \in N$:

$$\begin{aligned}\frac{\partial \text{err}}{\partial z^{\wedge}_{i,j}} &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \frac{\partial M(\mathbf{u})}{\partial w_i} \cdot \frac{\partial w_i}{\partial z_i} \\ &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot h^{\wedge}_{i,j} \cdot \sigma(z^{\wedge}_{i,j}) \\ \frac{\partial \text{err}}{\partial z^{\vee}_{i,j}} &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \frac{\partial M(\mathbf{u})}{\partial w_i} \cdot \frac{\partial w_i}{\partial z_i} \\ &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot h^{\vee}_{i,j} \cdot \sigma(z^{\wedge}_{i,j})\end{aligned}$$

7.3.7.3 Gradient w.r.t. u

Now that we have computed the gradient for the parameters Z, we can compute the gradient to be back-propagated to the modules that might be upstream of the current one. This is, in essence, the gradient of $\mathbf{u}$.

Due to the non-differentiable operations (namely, min and max poolings), the gradient is not defined everywhere w.r.t u_i. In particular, if $u_i = u_j$, $u_i \wedge u_j$ has no defined gradient (see Figure 7.4). We thus decide to use the following obious subgradient:

$$\frac{\partial(u_i \wedge u_j)}{\partial u_i} = \begin{cases} 1 \text{ if } u_i < u_j \\ 0 \text{ if } u_i > u_j \\ 0.5 \text{ if } u_i = u_j \end{cases}$$

and

$$\frac{\partial(u_i \vee u_j)}{\partial u_i} = \begin{cases} 0 \text{ if } u_i < u_j \\ 1 \text{ if } u_i > u_j \\ 0.5 \text{ if } u_i = u_j \end{cases}$$

We can now compute the gradient to backpropagate. $\forall i \in N$:

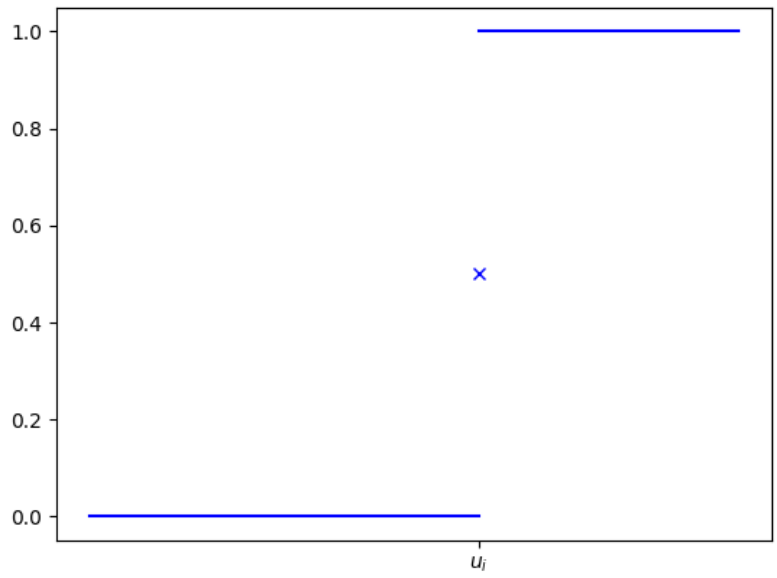


Figure 7.4: Plot of the subgradient $\frac{\partial(u_i \vee u_j)}{\partial u_i}$ as a function of u_j, with a fixed u_i

$$\begin{aligned}
\frac{\partial \text{err}}{\partial u_i} &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \frac{\partial M(\mathbf{u})}{\partial u_i} \\
&= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \left(w_i + \sum_{j \in N \setminus i} \left(w^{\wedge}_{i,j} \cdot \frac{\partial (u_i \wedge u_j)}{\partial u_i} + w^{\vee}_{i,j} \cdot \frac{\partial (u_i \vee u_j)}{\partial u_i} \right) \right) \\
&= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \left(w_i + \sum_{\substack{j \in N \setminus \{i\} \\ u_i < u_j}} w^{\wedge}_{i,j} + \sum_{\substack{j \in N \setminus \{i\} \\ u_i > u_j}} w^{\vee}_{i,j} + 0.5 \sum_{\substack{j \in N \setminus \{i\} \\ u_i = u_j}} (w^{\vee}_{i,j} + w^{\wedge}_{i,j}) \right)
\end{aligned}$$

7.4 3-additive 0-1-FM-based Choquet Integral

7.4.1 General Results

Now that we have tackled the learning of 2-additive Choquet integrals, we can address the learning to a more general class, that is, the 3-additive ones. Nonetheless, this leads to the following issues:

- there is no interpretable expression of the 3-additive CI
- there is an exponential number of monotonicity constraints

We would like to have an expression such as that from Equation (7.3), where the convex combinations of min and max operators is easily understable. As a consequence, we focus on a subset of the 3-additive CIs, which is described below.

Note that the following results can be found in [Grabisch, 2016]. We present them as they are necessary for understanding our implementation.

7.4.1.1 Convex Combinations of Fuzzy Measures

Property 2. *Let $\nu = \{\mu_1, ..., \mu_m\}$ a set of FMs on $N = \{1, ..., n\}$. Let μ_Σ the convex sum:*

$$\mu_\Sigma = \sum_{i=1}^{m} w_i \mu_i$$

with weights w non-negative and which sum to one. Then μ_Σ is also a valid FM. In particular, the associated Möbius values follow the same relation as μ, that is, $\forall S \subseteq N, m_{\mu_\Sigma}(S) = \sum_{k=1}^{m} w_k m_{\mu_k}(S)$.

Corollary 5. *If all FMs $\nu = \{\mu_1, ..., \mu_m\}$ are k-additive, then any convex sum of elements of ν is also k-additive.*

Corollary 6. *Given μ_Σ as described above, we have, for any $\mathbf{u} \in [0,1]^n$:*

$$C_{\mu_\Sigma}(\mathbf{u}) = \sum_{i=1}^{m} w_i C_{\mu_i}(\mathbf{u})$$

7.4.1.2 0-1 Fuzzy Measures

A *0-1 fuzzy measure* μ is a particular FM, which only takes values 0 and 1. Obviously, by **Monot**, we have that:

$$\forall A \subseteq N,\ \mu(A) = 1 \ \Rightarrow\ \forall B \supseteq A,\ \mu(B) = 1$$

and

$$\forall A \subseteq N,\ \mu(A) = 0 \ \Rightarrow\ \forall B \subseteq A,\ \mu(B) = 0$$

We now look at an interesting way of describing a 0-1-FM.

Definition 10. *An **antichain** on 2^N (the powerset of N) together with the inclusion relation $\subseteq$ is a set of parts of N $\mathcal{D} = \{S_1, ..., S_p\}$, with $\forall i \in \{1, p\}$, $S_i \subseteq N$, and there are no two $S_i, S_j \in \mathcal{D}$ such that $S_i \subseteq S_j$.*

Then, given a non-empty antichain $\mathcal{D}$, one can define its associated 0-1-FM $\mu_\mathcal{D}$ s.t., $\forall S \subseteq N$:

$$\mu_\mathcal{D}(S) = \begin{cases} 1 & \text{if } \exists\ B \in \mathcal{D},\ B \subseteq S \\ 0 & \text{otherwise} \end{cases}$$

Note that there is a bijection between the set of antichains and that of 0-1-FM for a given N.

Moreover, a CI parameterized by a 0-1 measure is easy to compute:

$$\begin{aligned} C_{\mu_\mathcal{D}}(\mathbf{u}) &= \sum_{i=1}^{n} u_{\tau(i)} \left(\mu_\mathcal{D}(A_i) - \mu_A(A_{i+1})\right) \\ &= \sum_{i=1}^{n} u_{\tau(i)} \left(\mathbb{1}_{\{\exists\ B\in\mathcal{D},\ B\subseteq A(i)\}} - \mathbb{1}_{\{\exists\ B\in\mathcal{D},\ B\subseteq A(i)\}}\}\right) \\ &= \bigvee_{S_k\in\mathcal{D}} \bigwedge_{i\in S_k} x_i \end{aligned} \tag{7.9}$$

7.4.1.3 0-1 Fuzzy Measures as a Base

We denote by $\mathcal{M}_k^{0-1}$ the set of k-additive 0-1-FM, and by $\mathcal{M}_k$ the set of k-additive FM.

Using the results from both sections above, we can now define a subclass $\mathcal{C}_k$ of k-additive FM, which are all convex combinations of the elements in $\mathcal{M}_k^{0-1}$. Formally:

$$\mathcal{C}_k = \left\{ \sum_{\mu_\mathcal{D}\in\mathcal{M}_3^{0-1}} w_\mathcal{D}\mu_\mathcal{D}\ ;\ \forall\ \mu_\mathcal{D} \in \mathcal{M}_k^{0-1},\ w_\mathcal{D} \geq 0 \text{ and } \sum_{\mu_\mathcal{D}\in\mathcal{M}_k^{0-1}} w_\mathcal{D} = 1 \right\} \tag{7.10}$$

We see that the monotonicity and normalization constraints are written as:

$$\forall\ \mu_\mathcal{D} \in \mathcal{M}_k^{0-1},\ w_\mathcal{D} \geq 0 \tag{7.11}$$

and

$$\sum_{\mu_\mathcal{D}\in\mathcal{M}_k^{0-1}} w_\mathcal{D} = 1 \tag{7.12}$$

Interestingly, $\mathcal{C}_2$ is exactly the set of 2-additive fuzzy measures; that is, $\mathcal{M}_2^{0-1}$ is a base of $\mathcal{M}_2$. This, though, is not true for $k \geq 3$. Nonetheless, as Corollary 5 states, we know that $\mathcal{C}_k \subseteq \mathcal{M}^k$ ($\mathcal{C}_k \subset \mathcal{M}^k$ if $k > 2$).

In practice, we wish to implement a neural module which allows to learn elements in $\mathcal{C}_3$. In order to do so, nonetheless, we need to first compute the base $\mathcal{M}_3^{0-1}$ of the search space. This comes down to being able to computing the set $\mathcal{D}_3^{0-1}$ of all of the antichains corresponding to all elements in $\mathcal{M}_3^{0-1}$.

Fortunately, they can all be expressed in a simple way. We list below the 10 forms that the elements in $\mathcal{D}_3^{0-1}$ can take:

1. $\{\{i\}\}$ for $i \in N$ (n elements)
2. $\{\{i,j\}\}$ for $\{i,j\} \subseteq N,\ i \neq j$ ($\frac{n(n-1)}{2}$ elements)
3. $\{\{i\},\{j\}\}$ for $\{i,j\} \subseteq N,\ i \neq j$ ($\frac{n(n-1)}{2}$ elements)
4. $\{\{i,j,k\}\}$ for $\{i,j,k\} \subseteq N, i \neq j \neq k$ ($\frac{n(n-1)(n-2)}{6}$ elements)
5. $\{\{i\},\{j\},\{k\}\}\{i,j,k\} \subseteq N, i \neq j \neq k$ ($\frac{n(n-1)(n-2)}{6}$ elements)
6. $\{\{i\},\{j,k\}\}$ for $\{i,j,k\} \subseteq N, i \neq j \neq k$ ($\frac{n(n-1)(n-2)}{2}$ elements)
7. $\{\{i,j\},\{j,k\}\}$ for $\{i,j,k\} \subseteq N, i \neq j \neq k$ ($\frac{n(n-1)(n-2)}{2}$ elements)
8. $\{\{i,j\},\{i,k\},\{j,k\}\}$ for $\{i,j,k\} \subseteq N, i \neq j \neq k$ ($\binom{n}{3}$ elements)
9. $\{\{i,j\},\{k,l\},\{i,l\}\}$ for $\{i,j,k,l\} \subseteq N, i \neq j \neq k$ ($2\binom{n}{2}\binom{n-2}{2}$ elements)
10. $\{\{i,j,k\},\{i,l\},\{j,l\},\{k,l\}\}$ for $\{i,j,k,l\} \subseteq N, i \neq j \neq k$ ($(n-3)\binom{n}{3}$ elements)

Due to an error in our initial identification of the antichains, we do not actually use the forms 8, 9 and 10 below. We thus, in practice, use only a subset of these antichains of shapes 1 to 7. We call below $\mathcal{D}_3^*$ this subset, and $\mathcal{M}_3^*$ the corresponding set of $0-1$ FMs.

Using these explicit forms, we can now generate any antichain in $\mathcal{D}_3^*$. Note that the first 3 forms correspond to the elements of $\mathcal{D}_2^{0-1}$. We can compute the total number of antichains:

$$\begin{aligned}
|\mathcal{D}_3^*| &= n + 2\frac{n(n-1)}{2} + 2\frac{n(n-1)(n-2)}{6} + 2\frac{n(n-1)(n-2)}{2} \\
&= n^2 + \frac{4}{3}n(n-1)(n-2) \\
&= \frac{4}{3}n^3 - 3n^2 + \frac{8}{3}n
\end{aligned}$$

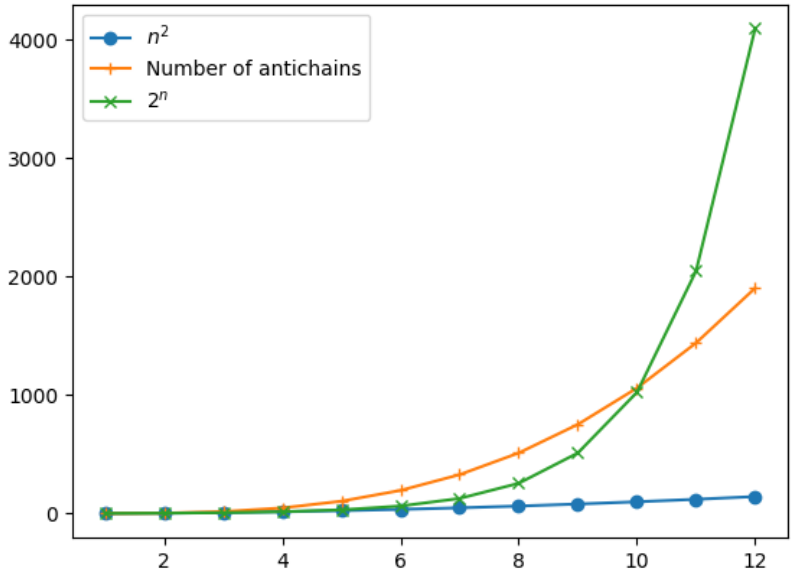


Figure 7.5: Comparing the complexity of the module depending on the number of criteria

While this is polynomial, it is important to note that it is still a large number of parameters. In particular, if the aggregator has fewer than 9 criteria, a $0-1$-based 3-additive CI has more parameters in this representation than a general CI, as is shown in Figure 7.5.

It is to be noted that, if we were counting all antichains in $\mathcal{D}_3^{0-1}$, the number of parameters would be $\mathcal{O}(n^4)$. Omitting shapes 9 and 10 allow this number to remain cubic. It would be interesting to study formally how much representativity is lost in this case. This could be done as a computation of volumes of both polytopes, or the maximal distance between a point μ in $\mathcal{M}_3^{0-1}$ and the point of $\mathcal{M}_3^*$ that is closest to μ.

Below, we use only the antichains of forms 1 to 7. We work with the set:

$$\mathcal{C}_3^* = \left\{ \sum_{\mu_\mathcal{D} \in \mathcal{M}_3^*} w_\mathcal{D} \mu_\mathcal{D} \ ; \ \forall \ \mu_\mathcal{D} \in \mathcal{M}_3^*, \ w_\mathcal{D} \geq 0 \text{ and } \sum_{\mu_\mathcal{D} \in \mathcal{M}_3^*} w_\mathcal{D} = 1 \right\} \tag{7.13}$$

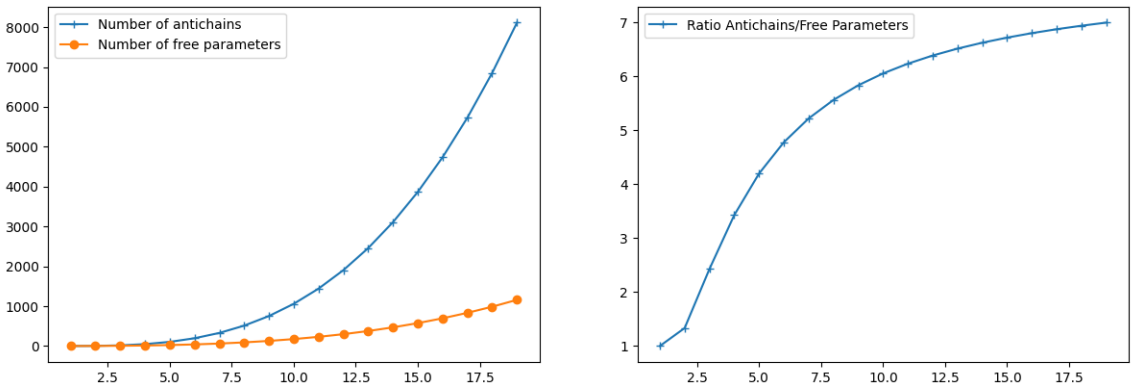


Figure 7.6: Comparing the number of antichains in $\mathcal{D}_3^*$ and of free Möbius values

7.4.2 Remark on the number of parameters

We recall that a fuzzy measure μ, with Möbius values m, is defined on all subsets of N. It thus has 2^n free (but constrained) parameters. In the case of a 3-additive FM, it is imposed by definition that, for each $A \subseteq N$ such that $|A| > 3$, we have $m(A) = 0$. Moreover, we have $m(\emptyset) = 0$.

As a consequence, the free parameters of a 3-additive FM are only the Möbius values of all $A \subseteq N$ such that $1 \leq |A| \leq 3$. There are, respectively:

- n subsets with 1 element
- $\frac{n(n-1)}{2}$ subsets with 2 elements
- $\frac{n(n-1)(n-2)}{6}$ subsets with 3 elements

The total number of free parameters is thus

$$\begin{aligned} n + \frac{n(n-1)}{2} + \frac{n(n-1)(n-2)}{6} &= \frac{1}{6}(6n + 3n^2 - 3n + n^3 - 3n^2 + 2n) \\ &= \frac{1}{6}(5n + n^3) \end{aligned}$$

The comparison of this value with the number of antichains (and thus of parameters in the antichain-based representation) can be found in Figure 7.6.

We see that there is a larger number of parameters in the antichain-based representation, with a ratio that converges to 8 as the number of criteria goes towards infinity.

	Parameters	**Monot** Complexity
3-additive	$\frac{1}{6}(5n + n^3)$	$n2^n - 1$
$\mathcal{C}_3^*$	$\frac{4}{3}n^3 - 3n^2 + \frac{8}{3}n$	$\frac{4}{3}n^3 - 3n^2 + \frac{8}{3}n$
General capacity	2^n	$n2^n - 1$

Table 7.2: Trade-off: Möbius-based vs weights-based parameterization

Nonetheless, this increase in the number parameter allows to write the set of constraints on each parameter much more easily. This is the same philosophy as used in 7.3 for the 2-additive FMs, where we roughly doubled the number of parameters in the representtaion in order to avoid an otherwise exponential number of parameters.

Table 7.2 shows the comparison between the complexity of the FMs in $\mathcal{C}_3^*$, the 3-additive FMs, and the general ones, in terms of parameters and monotonicity constraints.

7.4.3 Implementation as a Neural Module

We explain in this section how we implement a module which learns any function in $\mathcal{C}_3^*$, and only those. First of all, we define a neuron which implements a CI parameterized by a 0-1 3-additive FM. Then, we show how we assemble such modules in order to learn the final function.

7.4.3.1 Implementing a $0-1$-based 3-Additive Neuron

The special type of neurons we present here efficiently compute the output of a CI parameterized by $\mu \in \mathcal{M}_3^*$. Indeed, such CIs are the base component for elements of CIs parameterized by measures in $\mathcal{C}_3^*$.

Let $\mathcal{D}$ the antichain associated with μ. Then, given an input vector $\mathbf{u}$, the output $C_\mu(\mathbf{u})$ is computed following Algorithm 3.

Algorithm 3 Computing the Output of a CI with a 0-1 FM

sorted_indices $\leftarrow [(1), (2), ...(n)]$ s.t. $u_{(1)} \leq u_{(2)} \leq ... \leq u_{(n)}$
remaining_indices $= \{1, ..., n\}$
curr_ind $\leftarrow 1$
while $\exists S \in \mathcal{D}$ s.t. $S \subseteq$ remaining_indices **do**
    curr_ind $\leftarrow$ curr_ind+1
    remaining_indices $\leftarrow$ remaining_indices$\backslash\{$(index)$\}$
end while
passed_index $=$ curr_ind-1
return $u_{(\text{curr\_ind-1})}$

The complexity of the first line is in $\mathcal{O}(n \log(n))$, because of the need to sort. The *while* loop has at most n steps; at each step, we need to check whether:

$$\exists S \in \mathcal{D} \text{ s.t. } S \subseteq \text{ remaining\_indices}$$

Considering that checking $B \subseteq A$ has complexity $\mathcal{O}(|B|)$ if A is a set, the above line has complexity

$$\mathcal{O}\left(\sum_{S \in \mathcal{D}} |S|\right) \leq 4$$

It is thus reasonable to write that the **while** loop has complexity $\mathcal{O}(n)$. Thus, the evaluation of a given CI parameterized with $\mu \in \mathcal{M}_3^*$ has complexity $\mathcal{O}(n \log(n))$.

Note that we store the index of the element of **u** that was returned. While this is not useful at the moment, it will be during backpropagation.

7.4.3.2 Building the module

Now that we can efficiently evaluate a CI parameterized by any FM in $\mathcal{M}_3^*$, we can build a module which can represent any CI parameterized by an FM in $\mathcal{C}_3^*$.

In order to do that, we first build an n-dimensional input layer. Then we need a single hidden layer, composed of $m := |\mathcal{D}_3^*|$ hidden neurons $h_1, ..., h_m$, each implementing a CI parameterized by an FM in $\mathcal{M}_3^*$. Finally, we need a single output neuron. The weights between the input and hidden layer are all 1, and the weights between the hidden and the output layer are the weights of the convex sum defined in (7.13). This is illustrated in Figure 7.7.

Note that the number of neurons is cubic w.r.t. n; thus, as there is one weight for each hidden neuron, the number of free parameters of the model is cubic as well. The weights between the input layer and the hidden layer are always set to

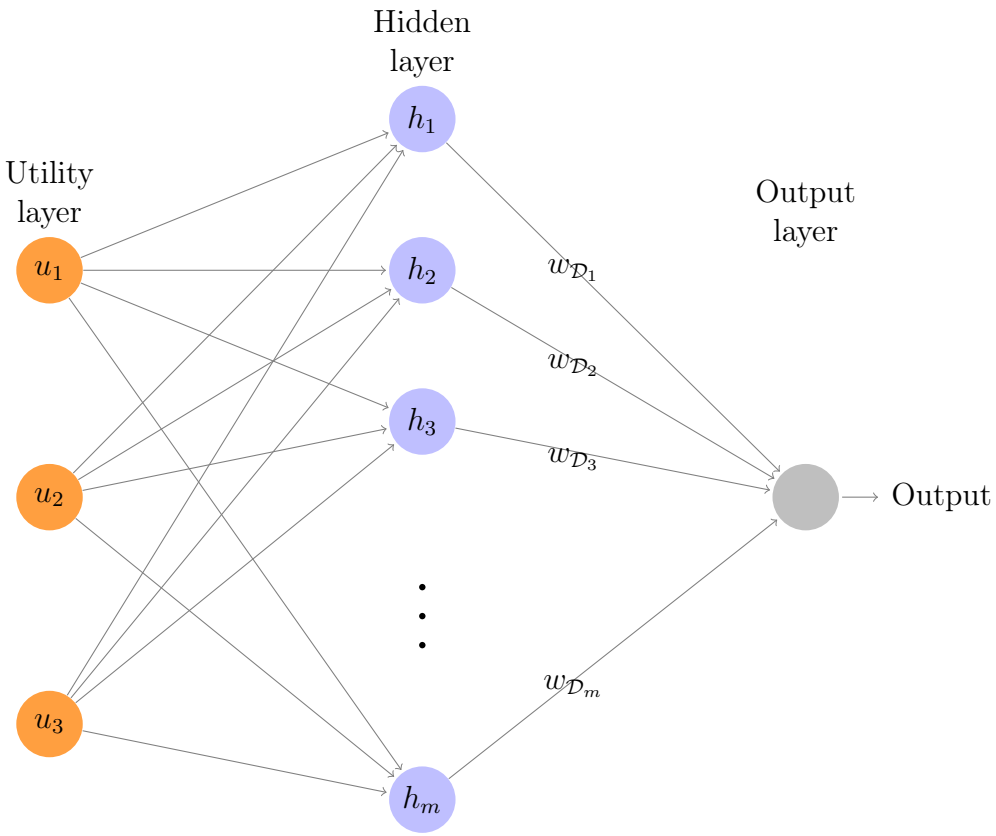


Figure 7.7: A 3-additive Choquet module with 3 inputs

1, and do not change during training, as the aim of this layer is simply to dispatch the input among all of the $0-1$ Choquet integrals represented by each hidden layer.

7.4.3.3 Forward Propagation

The forward propagation in such a module Q is made in two steps. Given the input $\mathbf{u} \in [0,1]$, we compute:

$$\begin{aligned} h(\mathbf{u}) &= (h_1(\mathbf{u}), ..., h_m(\mathbf{u})) \\ &= (C_{\mu_{\mathcal{D}_1}}(\mathbf{u}), ..., C_{\mu_{\mathcal{D}_m}}(\mathbf{u})) \end{aligned} \tag{7.14}$$

Thus, we have the evaluation of $\mathbf{u}$ by all 3-additive 0-1 CIs in $\mathcal{M}_3^*$. Note that, given the $\mathcal{O}(n^3)$ number of CIs, and the $\mathcal{O}(n\log(n))$ cost of evaluating a CI, yields a total cost of $\mathcal{O}(n^4\log(n))$ for computing $\mathrm{Q}(\mathbf{u})$. While rapidly increasing, this remains tractable for many applications.

Then exactly as in 7.13, we compute $W \bullet h(\mathbf{u})$, with $W = (w_{\mathcal{D}_1}, ..., w_{\mathcal{D}_m})$ the weight vector for the output node.

The output is thus:

$$W \bullet h(\mathbf{u}) = \sum_{\mu_{\mathcal{D}} \in \mathcal{M}_3^{0-1}} w_{\mathcal{D}} C_{\mu_{\mathcal{D}}}(\mathbf{u}) \tag{7.15}$$

which is an element of $\mathcal{C}_3^*$, given that W satisfies the right properties. We show now how to ensure the validity of W, and thus that of the model.

7.4.4 Validity of the Module

Theorem 11. *Let $\mathcal{E}_{\mathrm{C3}}^*$ be the domain of all functions that a 0-1 based 3-additive CI module can take, assuming that its weight vector satisfy* (7.11) *and* (7.12)*. Let also $\mathcal{E}_{\mathrm{C3}} = \{C_\mu,\ \mu \in \mathcal{C}_3^*\}$ the set of all CIs parameterized by an FM in $\mathcal{C}_3^*$.*

Then $\mathcal{E}_{\mathrm{C3}}^* = \mathcal{E}_{\mathrm{C3}}$.

That is, these modules can represent any function in $\mathcal{C}_3^*$, and only those. We can now prove this result.

Proof : **1.** $\mathcal{E}_{\mathrm{C3}}^* \subseteq \mathcal{E}_{\mathrm{C3}}$:
By (7.15), the output of a module Q is:

$$\mathrm{Q}(\mathbf{u}) = \sum_{\mu_{\mathcal{D}} \in \mathcal{M}_3^*} w_{\mathcal{D}} C_{\mu_{\mathcal{D}}}(\mathbf{u})$$

By Corollary 6, this means that Q ca be written as:

$$M(\mathbf{u}) = C_{\mu_\Sigma}(\mathbf{u}) \text{ with } \mu_\Sigma = \sum_{\mu_\mathcal{D} \in \mathcal{M}_3^*} w_\mathcal{D} \mu_\mathcal{D}$$

Thus $\mathrm{Q} \in \mathcal{E}_{\mathrm{C3}}$, which shows the first inclusion.

2. $\mathcal{E}_{\mathrm{C3}}^* \supseteq \mathcal{E}_{\mathrm{C3}}$:
Let $\mathcal{F} \in \mathcal{E}_{\mathrm{C3}}$. By definition, $\exists \mu \in \mathcal{C}_3^*$ s.t. $\mathcal{F} = C_\mu$ with $\mu = \sum_{\mu_\mathcal{D} \in \mathcal{M}_3^*} w_\mathcal{D} \mu_\mathcal{D}$

for a certain set of weights that satisfy (7.11) and (7.12). Using this set of weight to parameterize the output layer of a module Q yields, by (7.15), the result that $\mathrm{Q} = C_\mu$. Thus, we have $\mathcal{F} \in \mathcal{E}_{\mathrm{C3}}^*$.

This concludes the proof. □

7.4.5 Ensuring the Validity of the Constraints

As we have seen, in order for the set $\mathcal{E}_{\mathrm{C3}}^*$ of functions representable by such a network Q to be equal to $\mathcal{C}_3$, we need W to satisfy:

- $\forall\ \mu_\mathcal{D} \in \mathcal{M}_3^*,\ w_\mathcal{D} \geq 0$
- $\sum_{\mu_\mathcal{D} \in \mathcal{M}_3^*} w_\mathcal{D} = 1$

These conditions are ensured in exactly the same way as for the 2-additive module (see 7.3.6). That is, the $w_\mathcal{D}$ are computed as the softplus of latent variables $z_\mathcal{D}$ in order to ensure non-negativity. Moreover, the $w_\mathcal{D}$ are normalized at each epoch by being divided by their sum.

These two procedures ensure that the module always yields a valid function from $\mathcal{C}^3$. Nonetheless, as the softplus has its image in $\mathbb{R}_+^*$, the search space $\mathcal{E}_{\mathrm{C3}}^*$ is not exactly $\mathcal{E}_{\mathrm{C3}}$, but is a dense part; i.e. a module Q can approximate any function in $\mathcal{E}_{\mathrm{C3}}$ to an arbitrary precision.

7.4.5.1 Computing the gradients

Considering the highly linear nature of this aggregator, computing the gradients for each free parameter $z_\mathcal{D}$ is quite straightforward. Once again, given the gradient of the error err w.r.t. $\mathrm{Q}(\mathbf{u})$, we have:

$$\begin{aligned} \frac{\partial \mathrm{err}}{\partial z_\mathcal{D}} &= \frac{\partial \mathrm{err}}{\partial M(\mathbf{u})} \cdot \frac{\partial M(\mathbf{u})}{\partial w_\mathcal{D}} \cdot \frac{\partial w_\mathcal{D}}{\partial z_\mathcal{D}} \\ &= \frac{\partial \mathrm{err}}{\partial M(\mathbf{u})} \cdot h_\mathcal{D}(\mathbf{u}) \cdot \sigma(z_\mathcal{D}) \end{aligned} \tag{7.16}$$

As for the gradients to back-propagate, that is the derivative of err w.r.t. $\mathbf{u}$, we have to use the variable `passed_index` that we stored during forward-propagation (see Sec 7.4.3.1). For a given antichain $\mathcal{D}$, we write $\text{pass}_{\mathcal{D}}$ this variable, which is the index of the element of $\mathbf{u}$ that was returned by $h_{\mathcal{D}}$.

Thus, the gradient for u_i is computed as:

$$\begin{aligned}\frac{\partial \text{err}}{\partial u_i} &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \frac{\partial M(\mathbf{u})}{\partial h(\mathbf{u})} \cdot \frac{\partial h(\mathbf{u})}{\partial u_i} \\ &= \frac{\partial \text{err}}{\partial M(\mathbf{u})} \cdot \sum_{\mu_{\mathcal{D}} \in \mathcal{M}_3^{0-1}} w_{\mathcal{D}} \mathbb{1}_{\{i\}}(\text{pass}_{\mathcal{D}})\end{aligned} \tag{7.17}$$

That is, only the values that were actually forward-propagated get a gradient, the other ones get 0, as they did not influence the output (and thus the error).

CHAPTER 8

ASSEMBLING AND TRAINING A NETWORK

Contents

8.1 General Considerations

We have seen in the previous chapters how we could implement several different neural modules. Each of these modules implements a given type of function, either a marginal utility or a Choquet integral-based aggragator.

We have seen how to compute the output of each module given its input, and how to compute the gradient of each parameter given a backpropagated error. We have also seen how to ensure, for each module, that it satisfies all constraints that make it a valid instance if the function class it is supposed to represent.

In this chapter, we will present how to assemble these modules in order to build diverse, more complex models. Such models are presented below. While they are not the only thus buildable models, they present interesting properties, which we will describe and exploit.

First, we present the UHCI model, introduced in 2.7.4 and in which the aggregators are disposed in a tree-like manner. Then, we generalize this model to any directed acyclic graph-structure (DAGs). Finally, we present how we train the thus built models in different contexts, in order to deal with different types of data and to solve different problems.

8.2 Hierarchical Choquet Integral

8.2.1 Recalls

In this section, we show how to implement a module which can represent a hierarchical Choquet integral (HCI) using the previously defined aggregator module. The HCI is presented in Section 2.7.2.1. We recall that it can be defined by:

- a hierarchy $\mathcal{T} = \langle r, M, \mathrm{Ch} \rangle$, which is a rooted tree with:
  - n leaves $N = \{1, ..., n\}$ (one for each criterion, hence the abuse of notations of N)
  - non-leaf nodes V
  - a relation $\mathrm{Ch} : V \to 2^M$, with $M = V \cup N$, which, to each node k, associates the set of its children

Module	Inputs
Q_8	$(\text{out}_2,\ \text{out}_3)$
Q_9	$(\text{out}_1,\ \text{out}_8)$
Q_{10}	$(\text{out}_4,\ \text{out}_5,\ \text{out}_6)$
Q_{11}	$(\text{out}_9,\ \text{out}_{10},\ \text{out}_{11})$

Table 8.1: Caption

- a set $\nu = \{\mu_k,\ k \in V\}$ of fuzzy measures, such that μ_k parameterizes the CI associated to node k

Then, the output of an HCI $\mathcal{A}$ given an alternative $\mathbf{u}$ is computed recursively, starting from the leaves, as:

$$a_k = \begin{cases} u_k & \text{if } k \in N \\ C_{\mu_k}((a_i,\ i \in \text{Ch}(k)) & \text{if } k \in V \end{cases}$$

We introduce a new notation, $\text{Pa} : M \setminus \{r\} \to V$, such that $\text{Pa}(k)$ is the parent node of k. That is: $\text{Pa}(k) = j \iff k \in \text{Ch}(j)$.

Now that this is re-established, we can proceed and assemble a network.

8.2.2 Architecture

Let $\mathcal{T} = \langle r, M, \text{Ch}\rangle$ a hierarchy on the set of criteria N. We still write $V = M \setminus N$. An HCI network on $\mathcal{T}$ is composed of $|V|$ Choquet neural modules $\{Q_k,\ k \in V\}$. We denote by inp_k the input layer of Q_k, and by out_k its output layer (reduced to a single neuron). For simplicity in the following, we also write for $k \in N$, $\text{out}_k = \text{inp}_k$, the input (leaves) nodes of the network.

Then, for $k \in V$, $|\text{inp}_k| = |\text{Ch}(k)|$. In particular, $\text{inp}_k = \{\text{out}_i,\ i \in \text{Ch}(k)\}$; that is, the output neuron of a given module belongs to the input layer of the module that represents its parent in $\mathcal{T}$.

Note that the Q_k can be of any type that is implemented by a module. They do not need to all be of the same type; 2-additive, general and 0-1-based 3-additive modules can be mixed according to the expertise of the network designer.

Example 29 (Ex. 1 cont'd)**.** *Assume the hierarchy from the previous examples (Figure 2.2). As we have 5 aggregation nodes (with indices 8 to 11), we will need 4 Choquet modules to represent it as an HCI network. The inputs of each module are given in table 8.1.*

We illustrate in Figures 8.2 a 2-additive HCI network on a small hierarchy given in 8.1, with three criteria and two internal nodes. Then figure 8.3 illustrates a 2-additive HCI network on the hierarchy from Example 1. We chose 2-additive CIs for simplicity, but any aggregator module could be used.

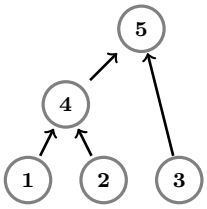


Figure 8.1: Example hierarchy, with 3 criteria and 2 aggregations.

8.2.3 Forward and Backpropagation

8.2.3.1 Forward-Propagation

Given a tree $\mathcal{T} = \langle r, M, \mathrm{Ch} \rangle$ and a set of FM $\nu = \{\mu_k,\ k \in V\}$ with n leaves, and an HCI network H defined on $\mathcal{T}$ with FM ν as described in the previous section, we compute the output of the network recursively, as described in Algorithm 4.

Algorithm 4 Forward propagation with input $\mathbf{u}$

Input: $\mathbf{u} \in [0,1]^n$
Function: forward_node$(k, \mathbf{u})$:
  if $k \in N$ **then**
    $\mathrm{out}_k \leftarrow u_k$
  else
    for $c \in \mathrm{Ch}(k)$ **do**
      forward_node$(c, \mathbf{u})$
    end for
    $\mathrm{out}_k \leftarrow C_{\mu_k}(\mathrm{out}_c \text{ for } c \in \mathrm{Ch}(k))$ ▷ i.e. computing the output of the module
  end if
end Function

forward_node(r, $\mathbf{u}$)
Return out_r

This allows to compute the output of the modules by exploiting the tree structure. All modules are computed, in the right order (that is, a module is only computed after all of its descendants have already yielded their own output).

Thus, given an HCI network H and a vector $\mathbf{u} \in [0,1]^n$, Algorithm 4 returns $H(\mathbf{u})$.

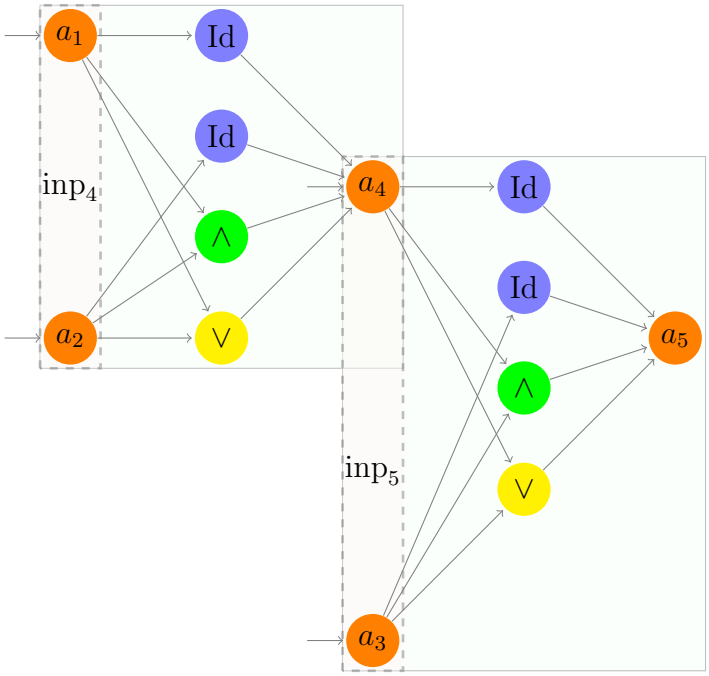


Figure 8.2: Example of the combination of two 2-additive modules on the hierarchy from Figure 8.1. Each module Q_4 and Q_5 is delimited by a box. The dashed boxes represent the input layer of their respective module. Each node with a free in-going arrow a_1 to a_3 is an input node of the module.

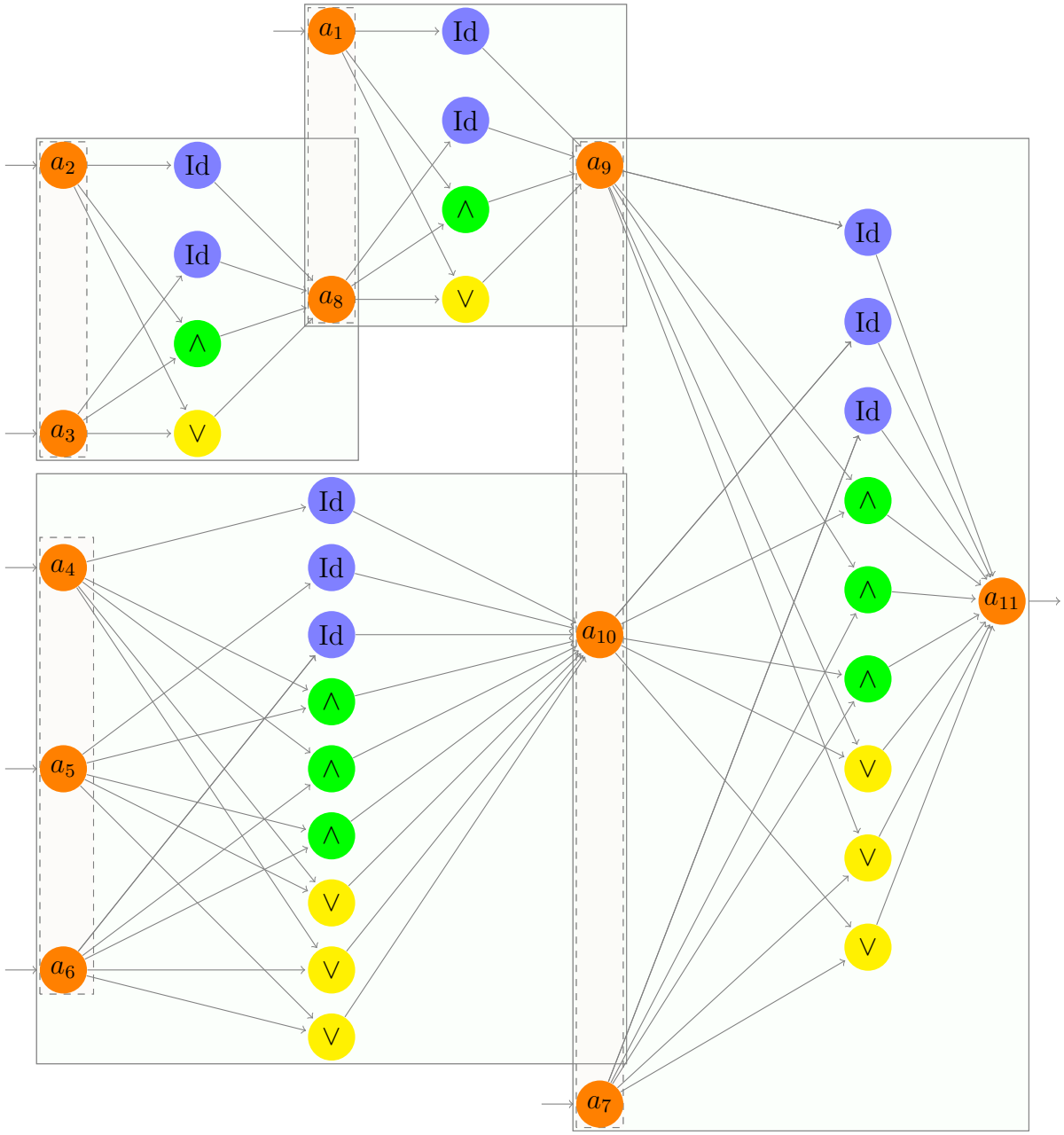


Figure 8.3: Example of the combination of two 2-additive modules on the hierarchy from Example 1. Each module is delimited by a box. The dashed boxes denote the input layer of each module. Each node with an in-going free arrow (a_1 through a_7) is an input node.

8.2.3.2 Backpropagation

The backpropagation algorithm, similarly, explores the network, starting from the root r, and following the Ch relation. This procedure is presented in Algorithm 5.

Algorithm 5 Gradient backpropagation

Input: $\frac{\partial \text{err}}{\partial H(\mathbf{u})}$
Function: compute_gradients(k, $\frac{\partial \text{err}}{\partial \text{Q}_k(\mathbf{u})}$):
  if $k \in V$ **then**
    Compute $\frac{\partial \text{err}}{\partial \theta}$ for any internal parameter θ of Q_k
    grad_input $\leftarrow \frac{\partial \text{err}}{\partial \text{inp}_k}$ ▷ *grad_input is composed of* $\{\frac{\partial \text{err}}{\partial \text{Q}_c(\mathbf{u})},\ c \in \text{Ch}(k)\}$
    for $c \in \text{Ch}(k)$ **do**
      compute_gradients(c, $\frac{\partial \text{err}}{\partial \text{Q}_c(\mathbf{u})}$)
    end for
  end if
end Function

compute_gradients(r, $\frac{\partial \text{err}}{\partial H(\mathbf{u})}$)

Once Algorithm 5 has been applied, all of the modules have computed the gradients of their parameters. They can then be used to update the parameters of the whole network, which is then ready for a new forward pass.

8.2.4 Showing the Validity of the Network

We show now that such an HCI network can represent any HCI with hierarchy $\mathcal{T}$ and with the right aggregator types. We work only with the aggregators that we have described and implemented before, that is:

1. C2: the 2-additive Choquet Integrals
2. C3: the 0-1-based 3-additive Choquet Integrals
3. CG: the General Choquet Integrals

We write $\mathcal{M}_{\text{C2}}$, $\mathcal{M}_{\text{C3}}$ and $\mathcal{M}_{\text{CG}}$ respectively the sets of 2-additive FMs, 0-1-based 3-additive FMs and general FMs. We extend the function type from Section 6.5.4 to yield the type of an aggregator (instead of only managing marginal utility modules).

$$\text{type} : M \to \{\text{NI}, \text{ND}, \text{SP}, \text{SV}, \text{C2}, \text{C3}, \text{CG}\}$$

which yields the type of a module given its corresponding node k in the hierarchy $\mathcal{T}$. Note that it is the type of the module, and not that of the function represented. That is, as $\mathcal{M}_{\text{C2}} \subseteq \mathcal{M}_{\text{C3}} \subseteq \mathcal{M}_{\text{CG}}$, it is possible that a module Q_k with type CG be parameterized to represent a 2-additive or 3-additive CI. Nonetheless, type(k) remains CG.

Then, we give the following representation theorem:

Theorem 12. *Let $\mathcal{T} = \langle r, M = V \cup N, \text{Ch} \rangle$. Let H an HCI module on $\mathcal{T}$, without restriction on the type of each aggregator. We call $\mathcal{E}^*$ the set of all possible functions that H can represent, by varying the parameters of all the aggregators while respecting the validity constraints of the aggregators.*

Then, we call $\mathcal{E}$ the set of all possible HCI functions defined on $\mathcal{T}$, and such that,

$$\forall k \in V,\ \mu_k \in \mathcal{M}_{\text{type}(k)}$$

Then $\mathcal{E}^*$ is a dense part of $\mathcal{E}$.

Theorem 12 tells that an HCI network H can represent any HCI with the same hierarchy whose aggregators are of the same type than those of H. It also states that it cannot represent any other HCI on hierarchy $\mathcal{T}$.

Proof : 1. $\mathcal{E}^*$ is dense in $\mathcal{E}$

Let $\mathcal{A} \in \mathcal{E}$. By Theorems 9, 10 and 11, each of the modules that compose H can approximate of any function of their respective type to an arbitrary precision. That is, regardless of the fuzzy measure $\mu_k \in \mathcal{M}_{\text{type}(k)}$, Q_k can represent C_{μ_k}. Thus, H can represent any $\mathcal{A} \in \mathcal{E}$ as accurately as possible, hence the density.

2. $\mathcal{E} \supseteq \mathcal{E}^*$

Let $H \in \mathcal{E}^*$. By Theorems 9, 10 and 11 each of the modules that compose H can only ever take the form of a function of their respective type. Thus, H is a an HCI with the right type of CI at any node; thus $H \in \mathcal{E}^*$.

The proof is completed. □

8.3 Directed Acyclic Graphs

Tree-like architectures have many qualities that make them suitable for decision models. Nonetheless, one might want to generalize them to not only trees (where each node has a single parent), but more general directed acyclic graphs, or DAGs.

We study, in this section, a generalization of HCI with a DAG-based structure. We present their implementation, and compare their advantages and drawbacks with regards to trees. We still call $N = \{1, ..., n\}$ the set of criteria.

8.3.1 Definition

We define the important notions that will be used in this section:

Definition 11. *A **Directed Graph** G is formed by a set of vertices M and a partial order relation* $\mathrm{Ch} : M \to \mathcal{P}(M)$. *An edge from vertex m_1 to vertex m_2 exists if and only if $m_2 \in \mathrm{Ch}(m_1)$. We say that m_2 is a child of m_1.*

We also define the opposite relation $\mathrm{Pa} : M \to \mathcal{P}(M)$, *such that,* $\forall (m_1, m_2) \in M,\ m_1 \in \mathrm{Pa}(m_2) \iff m_2 \in \mathrm{Ch}(m_1)$.

Definition 12. *A **path** in a directed graph G is a sequence of vertices $v_1, ..., v_m$ such that $\forall i \in \{1, ..., m\},\ v_{i+1} \in \mathrm{Ch}(v_i)$. The length of such a path is m, the number of edges in the path.*

Definition 13. *A **cycle** is a path $\pi = (v_1, ..., v_m)$, with length(π)> 1 such that $v_1 = v_m$, and all v_i with $i \in \{2, ..., m-1\}$ appear only once in π.*

Definition 14. *A **Directed Acyclic Graph** (DAG), is a directed graph that contains no cycle.*

In the reminder of this section, we will only work with DAGs that have suitable properties for being used as the structure of a neural network. That is, any DAG G used in this section will also satisfy the following properties:

- G is connected
- $\exists!\ v \in V$ such that $\mathrm{Pa}(v) = \emptyset$. We call this vertice r, or the *root*.
- Let $L = \{v \in V,\ \mathrm{Ch}(v) = \emptyset\}$. Then we enforce $|L| = n$, and we call the elements of L the *leaves*. By abuse of notation, we write $L = N$, as there is exactly one leaf for each criterion.
- $\forall v \in V,\ |\mathrm{Ch}(v)| \geq 2$

With these constraints, we see that the only difference between such a DAG G and a tree hierarchy H as defined before is that a vertice can now have several parents instead of a single one. The tree hierarchy is a special case of DAG where $\forall v \in M \setminus \{r\},\ \mathrm{Pa}(v) = 1$.

We use from now on the same notations that we used on trees in the previous section.

8.3.2 Implementation of a DAG-CI Network

We have seen in section 8.2.2 how to implement an HCI network. The implementation of a DAG-CI network is quite similar. Let $G = \langle r, M = V \cup N, \mathrm{Ch} \rangle$, a DAG. Then, for each $v \in V$, we build the aggregator module Q_v, with $\mathrm{inp}_v = \{\mathrm{out}_c,\ c \in \mathrm{Ch}(v)\}$.

8.3.3 Interests and Drawbacks

8.3.3.1 Advantages w.r.t. Trees

Directed acyclic graphs are an obvious generalization from tree structures. As a consequence, they offer much more expressivity. In particular, while the number of aggregations in a tree is bounded, a DAG can be arbitrarily large. We recall, from Section 5.1, that this allows to represent any non-decreasing, positively homogeneous, piecewise-linear functions.

At the same time, as we are working with Choquet integrals as aggregator, we ensure that the decision model remains monotonic (non-decreasing) and normalized. In particular, such interesting properties as idempotency and compensatoriness are preserved.

Given enough CIs, this model can thus approximate a very large set of functions, all constrained to the desired extent.

8.3.3.2 Drawbacks

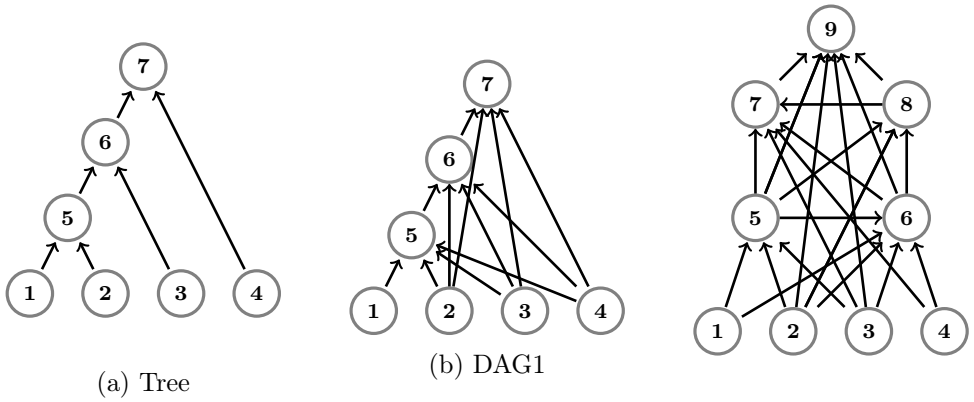
Nonetheless, this increase in expressivity comes at a cost. While the number of parameters in a tree-like structure is bounded w.r.t n (especially if the aggregator have limited additivity), there can be an arbitrarily large number of parameters for a DAG. This comes mostly from the fact that we can have arbitrarily many aggregators in a DAG, and that any vertex can have arbitrarily many ancestors.

Example 30. *To illustrate this, we use the DAGs from figure 8.4. For each model, we compute the number of parameters they would have, depending on the type of aggregator module, among 2-additive, 3-additive, and general Choquet Integral (we assume all aggregators have the same type, for simplicity).*

These numbers are given in Table 8.2.

This large number of parameters brings difficulty for training the model.

First of all, and most obviously, the computational cost of evaluating the model and computing the gradients can quickly explode, making training slow, or even intractable. Moreover, there is a much higher risk of overfitting, in particular with small training datasets.



(a) Tree

(b) DAG1

(c) DAG2

Figure 8.4: Example hierarchies, with 4 criteria

Graph	Inputs per Aggregations	2-additive	0-1-based 3-additive	General
Flat	4	16	48	16
Tree	2,2,2	12	12	12
DAG1	4,4,4	48	144	48
DAG2	3,5,5,3,6	104	440	144

Table 8.2: Number of parameters for the graphs given in Figure 8.1. The columns give the type of Choquet integral assumed. The fact that the general has fewer parameters comes from the fact that the aggregators are rather small on this example (see Figure 7.5)

An added difficulty is that this structure makes interpreting the model significantly harder, or even impossible. Indeed, a single criterion can go through many aggregators, and thus interact in many different ways with a large number of native and intermediate criteria. The thus obtained behaviours might prove too complex and numerous for en expert to interpret even for small DAGs.

Finally, there is no identifiability result on such DAG-CI (i.e. a composition of CI along a DAG strcture). In particular, due to the fact that we can create as many "clone" nodes (i.e. with the same Ch and Pa sets), it seems clear that the constraints to put on the structure and parameterization in order to obtain uniqueness would be highly restricting.

8.3.3.3 Discussion

The DAG-CI is thus a suitable model when interpretability is not of the essence, but we still wish our model to satisfy constraints such as monotonicity, idempotency or normalization. It can thus be seen as a constrained black-box model,

more regularized than a classic Multilayer Perceptron, but less so than an HCI.

In order to tackle the overfitting and computation cost issues, the three main axis on which we can act are:

- limit the number of aggregators, both in width and in depth
- limit the number of children of each vertice (and thus the dimension of each aggregator)
- limit the complexity of each aggregator, by employing lower-additivity modules

Note that the second and third options are the most efficient in limiting the number of parameters. Indeed, the number of parameters is:

- linear w.r.t. the number of nodes
- quadratic (2-additive), cubic (3-additive) or exponential (general) w.r.t. the size of the aggregators
- variable by orders of magnitude w.r.t. complexity of each aggregator

As a consequence, such a model should be used for representing complex yet constrained behaviours, with precautions taken as to how large the model can be.

8.3.4 Forward Propagation and Backpropagation

The forward propagation algorithm is quite similar to that of the Tree-based network. Starting at the root, we explore the tree in a depth-first manner, until we reach a leaf, and then compute the output of each module after all of its children have been computed.

The difference with a tree structure is that, if we applied the exact same algorithm, we would compute the output of a module as many times as this module has parents (as each parent would call it). This is obviously suboptimal, as these many computations would yield the same result.

As a consequence, we need to keep track of all the modules whose outputs have already been computed at this step. We thus use a set to do so, as shown in algorithm 6.

Backpropagation, now, is slightly more complex. Indeed, as a node can have several parents, the gradients from each parent must be summed. Thus, we can only treat a node if all of its ancestors have beed treated before; we keep track of the untreated nodes in a queue (called *to_treat* thereafter). This is illustrated in algorithm 7.

Algorithm 6 Forward propagation in a DAG-CI network, with input $\mathbf{u}$

Input: $\mathbf{u} \in [0,1]^n$
Function: forward_node($k, \mathbf{u}$, already_computed):
  already_computed $\leftarrow$ already_computed $\cup\{k\}$
  if $k \in N$ **then**
    $\text{out}_k \leftarrow u_k$
  else
    for $c \in \text{Ch}(k)$ **do**
      if $c \notin$ already_computed **then**
        forward_node($c, \mathbf{u}$, already_computed)
      end if
    end for
    $\text{out}_k \leftarrow C_{\mu_k}(\text{out}_c \text{ for } c \in \text{Ch}(k))$
  end if
end Function

already_computed $\leftarrow \{\}$
forward_node(r, $\mathbf{u}$, already_computed)
Return out_r

Algorithm 7 Gradient backpropagation in a DAG-CI

Input: $\frac{\partial \text{err}}{\partial H(\mathbf{u})}$
Function: compute_gradients(k, to_treat):
  if $\text{Pa}(k) \cap$ to_treat $= \emptyset$ **then** ▷ i.e. all ancestors have been treated
    to_treat $\leftarrow$ to_treat $\setminus\{k\}$
    $\frac{\partial \text{err}}{\partial \text{Q}_k(\mathbf{u})} \leftarrow \sum_{p \in \text{Pa}(k)} \frac{\partial \text{err}}{\partial \text{Q}_p(\mathbf{u})} \cdot \frac{\partial \text{Q}_p(\mathbf{u})}{\partial \text{Q}_k(\mathbf{u})}$
    Compute $\frac{\partial \text{err}}{\partial \theta}$
    grad_input $\leftarrow \frac{\partial \text{err}}{\partial \text{inp}_k}$
  else
    add k to the end of to_treat
  end if
  $v = \text{pop}(\text{to\_treat})$ ▷ v takes the first value in to_treat, it is removed it
  compute_gradients(v, to_treat)
end Function
compute_gradients(r, queue(V))

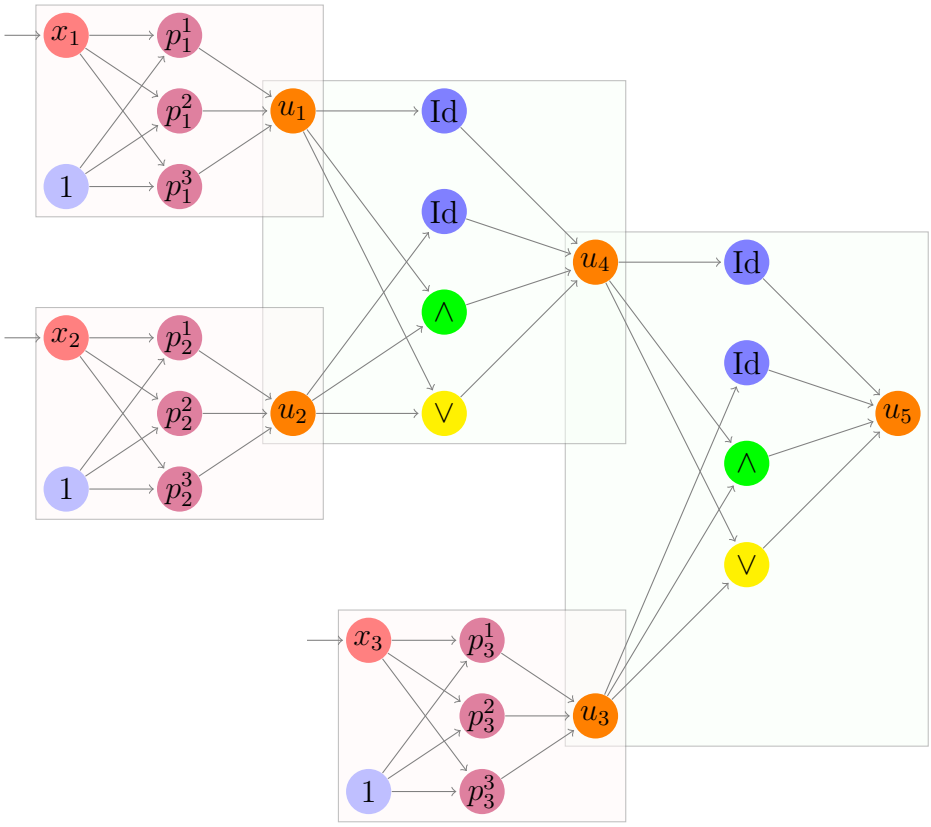


Figure 8.5: Example of the combination of the two modules on a tree with three monotonic marginal utility functions and two aggregation nodes.

8.4 Adding Marginal Utilities

We have shown in the previous sections how to implement neural networks to represent Choquet integral-based aggregators, with hierarchical structures such as trees and directed acyclic graphs.

A simple generalization is now to allow the models to represent UHCIs with such structures; that is, we exploit the marginal utility modules that were presented in Chapter 6. In order to do so, we generate n marginal utility modules, of any of the types implemented (including Selector), and insert them at the input nodes of an HCI or DAG-CI network. That is, the output of a given marginal utility module must be an input node of the network of CIs. This is illustrated in Figure 8.5 on a small hierarchical network with 3 criteria and 2 CIs.

Theorem 13. *Let H an HCI network with n inputs and hierarchy $\mathcal{T} = \langle r, M = V \cup N, \mathrm{Ch} \rangle$. Let $\mathcal{U} = \{\mathrm{Q}_1, ..., \mathrm{Q}_n\}$ a set of n marginal utility networks. The network H' is obtained by plugging the output node of each Q_i on an input node of H, as*

described earlier in this section.

Let $\mathcal{E}$ the set of UHCIs such that $\forall v \in V$, the aggregator is of type $\text{type}(v)$*, and $\forall k \in N$, the marginal utility u_k has one of the types reachable by a module of type* $\text{type}(k)$*.*

Let $\mathcal{E}^*$ the set of all functions representable by H' after training. Then $\mathcal{E}^*$ is dense in $\mathcal{E}$.

Proof : Theorem 12 tells us that the aggregator network is a dense part of the space of HCIs with hierarchy $\mathcal{T}$ with the right types of aggregators.

Theorems 4, 5 and 7, along with their respective corollaries, give us that the set of all functions representable by a module $\text{Q}_k \in \mathcal{U}$ is a dense part of the space of the types reachable by a module of type $\text{type}(k)$.

Combining such utility modules and such an aggregator network thus ensures that the whole network is a valid UHCI with the sought function type at each node. It also ensures that any such UHCI can be approximated to any extent, thus ending the proof. □

Note that, in any case, a value that is forward-propagated through the network will only go through a single marginal utility module; thus, it will only go through a single layer with sigmoid activations. This means that we do not suffer from the tendency of sigmoids to provoke a vanishing-gradients-phenomenon, which is well known in neural network with several layers of sigmoid-activated neurons.

8.5 Training Settings

8.5.1 End-to-End Training

We have seen in the above sections how to implement a fully valid Choquet-integral-based neural network, with a DAG-based structure and marginal utilities on all criteria. We have also seen how to compute their gradients from that of an error, or loss function.

We will now see how to train these networks in practice. NEUR-HCI can be used to train in three different settings. Those are, respectively, *Regression* (see Section 8.5.2), where the aim is to learn the precise, real satisfaction value, *Ordinal Regression* (see Section 8.5.4), where the data is to be classified into ordered satisfaction classes, and *Pairwise Preference Learning* (see Section 8.5.4), where the alternatives are given as pairs, with the information of which alternative is preferred to the other.

In those three settings, the preference model learned has the same type, that is, a valid UHCI or U-DAG-CI. In all sections below, we call H our network built from the modules above. We assume that, at all epochs, or even at each

Surface m^2	Garden m^2	Garage (yes/no)	Road km	Transp. km	Downtown km	Price €	Score
50	100	No	0.1	0.	0.	400,000	0.4
110	150	Yes	0.5	3.	4.	500,000	0.1
150	0	No	1.	0.5	0.5	450,000	0.9
150	30	No	0.1	5.	3.	300,000	0.1
500	1000	Yes	5.	5.	10.	1,500,000	0.3

Table 8.3: Caption

parameter update, each module is regularized through the procedures that ensure that it remains a valid instance of the functions it is supposed to represent. The procedure for a given type of module are described in the same section where the module's implementation is described (chapter 6 for marginal utilities, and 7 for aggregators).

We also call, below, DS the set of training data point, with $m := |\text{DS}|$, and err the immediate (i.e. example-wise) loss function.

8.5.2 Regression

The first and easiest setting to train in is regression. In this setting, the training is made on a dataset of the form:

$$\text{DS} = \{(\mathbf{x}^{(j)}, y^{(j)}), j = 1 \dots m\}$$

Example 31. *We present some houses in Table 8.4, defined on the criteria from example 1. The label for each house (i.e. its satisfaction score) is given in the last column.*

Each element in DS is a pair $(\mathbf{x}, y) \in X \times [0, 1]$. $\mathbf{x}$ is an alternative, and y is its expected score, normalized between 0 and 1. The aim of training will be for the model to predict scores as close as possible to the expected one.

This type of data would be rather difficult to collect in a preference context, as it implies that a decision maker can accurately label data, and give a satisfaction score that is as accurate as possible, which would be rather imprecise due to the subjectivity of that problem. This setting, nonetheless, becomes interesting and believable when the score (or value of interest), is the result of a physical measure or observation.

Example 32. *If the alternatives are computer parts (graphics processing unit), defined by three criteria :*

- *memory*
- *power consumption*
- *clock speed*

and we wish to score them, then we can use the number of rendered frames per seconds as an objective and accurate measurement of their performance (which can be normalized in the unit interval in order to have a score between zero and one).

A usual loss function for this setting is the mean squared error, that is, we aim at minimizing the total error:

$$\text{Err}(\text{DS}) = \frac{1}{m} \sum_{(\mathbf{X},y)\in\text{DS}} \text{err}(\mathbf{x}, y) := \frac{1}{m} \sum_{(\mathbf{X},y)\in\text{DS}} (H(\mathbf{x}) - y)^2$$

This loss is commonly used for such regression problems. Its gradient is easily computed. Given DS, the gradient of the error w.r.t. the output of the model (noted $H(\text{DS})$ by abuse of notation) can be computed as:

$$\begin{aligned}
\frac{\partial \text{Err}(\text{DS})}{\partial H(\text{DS})} &= \frac{1}{m} \sum_{(\mathbf{X},y)\in\text{DS}} \frac{\partial \text{err}(\mathbf{x}, y)}{\partial H(\mathbf{x})} \\
&= \frac{1}{m} \sum_{(\mathbf{X},y)\in\text{DS}} \frac{\partial (H(\mathbf{x}) - y)^2}{\partial H(\mathbf{x})} \\
&= \frac{2}{m} \sum_{(\mathbf{X},y)\in\text{DS}} (H(\mathbf{x}) - y)
\end{aligned}$$

As we have a closed-form expression for this gradient, we can now back-propagate through all of the modules, using the gradients given for each module.

8.5.3 Ordinal Regression

The second and setting to train in is ordinal regression. In this setting, the training is made on a dataset of the form:

$$\text{DS} = \{(\mathbf{x}^{(j)}, y^{(j)}), j \in \{1 \dots m\}\}$$

Each element in DS is a pair $(\mathbf{x}, y) \in X \times [0, 1]$. $\mathbf{x}$ is an alternative, and $y \in \{1, ..., K\}$ is its class of satisfaction. That is, we assume that there is a preference ordering of the classes, such that an element of class i is prefered to one of class $i - 1$.

Surface m^2	Garden m^2	Garage (yes/no)	Road km	Transp. km	Downtown km	Price €	Score
50	100	No	0.1	0.	0.	400,000	Average
110	150	Yes	0.5	3.	4.	500,000	Bad
150	0	No	1.	0.5	0.5	450,000	Good
150	30	No	0.1	5.	3.	300,000	Bad
500	1000	Yes	5.	5.	10.	1,500,000	Average

Table 8.4: Caption

The objective is, as in classification, to allow the model to label new examples with the right class. It is a problem that is very close to the preference learning problem called *instance ranking*, presented in Section 3.8.1.

Example 33. *We classify the houses in Table 8.4, defined on the criteria from example 1. The label (or class) is given in the last column. We assume 3 satisfaction classes : Bad, Average and Good, which are clearly ordered from least preferred to most preferred.*

This data is relatively easier to obtain than regression data, due to its lower accuracy. We can consider that a human would be able to indicate the correct class of satisfaction to associate to a given alternative, with high confidence (in particular in usual cases with two or three classes). These can also be obtained from implicit labels (for instance, if a user has "liked" or "disliked" some online content, this content can be classified as "good" or "bad" for the given user). As a consequence, behavioral data such as that used in recommender systems could be exploitable in this setting.

8.5.3.1 2 Classes

We first look at the basic case, where there are only two classes (i.e. "Good" and "Bad"). In this case, we can consider those labels to hold the respective values 1 and 0. An adapted loss for this problem is the logistic loss, which is heavily used in logistic regression and, later, choquistic utilitaristic regression [Tehrani et al., 2014].

The idea is to fit our model with a logistic sigmoid at its output node. That is, given H a UHCI or U-DAG-CI network, which returns a score $H(\mathbf{x})$ on alternative $\mathbf{x}$. Then, the probability that $\mathbf{x}$ belongs to class 1 is:

$$P(y = 1|\mathbf{x}) = \sigma_{\beta,\gamma}(H(\mathbf{x}))$$

The idea behind this is that the probability of $\mathbf{x}$ belonging to class 1 increases with $H(\mathbf{x})$, following a logistic sigmoid centered in β and with steepness parameter

γ. These two parameters are to be learned during training. The logistic loss on a dataset DS is given by:

$$\text{Err}(\text{DS}) = \sum_{(\mathbf{x},y)\in\text{DS}} [y \log(\sigma_{\beta,\gamma}(H(\mathbf{x}))) + (1-y)\log(1-\sigma_{\beta,\gamma}(H(\mathbf{x})))] \tag{8.1}$$

8.5.3.2 K classes

The binary classification problem can be generalized to more classes. All classes are denoted by the integers from 0 to $K-1$, with 0 being the least preferred class. There needs to be a preference order among the classes; that is, all alternatives from class $i \in \{0, 1, ..., K-1\}$ are preferred to all alternatives from class $j < i$.

Once again, we want the UHCI or U-DAG-CI to reflect that preference; that is, we want the class returned to be non-decreasing w.r.t. the model (a higher score will yield a more preferred class). While a sum of K step functions (a.k.a. Heaviside functions) would be formally valid, we need a differentiable function in order to have a gradient to back-propagate. Thus, we use a sum of $K-1$ logistic sigmoids $\sigma_{\beta_i,\gamma_i}$, with $i \in \{1, K-1\}$, in order to convert the score $H(x)$ into a predicted class. This is illustrated in Figure 8.6.

We can then use the mean squared error for training:

$$\text{err}(x) = \left(\sum_{i\in\{1,...,K-1\}} \sigma_{\beta_i,\gamma_i}(H(x)) - y \right)^2$$

For simplicity, we assume that the β_i are sorted, that is, $\beta_i \leq \beta_{i+1}$. This is not a strong constraint, as it can be done by re-indexing during or after training without changing anything to the model.

In order to do inference on the class of a given alternative $\mathbf{x}$, after training, we define a set of $K+1$ thresholds $t_i,\ i \in \{0, ..., K\}$, such that:

$$t_0 = 0,\ t_K = 1,\ t_i = \beta_i \text{ otherwise}$$

After training, the inference of a given class can be done as such:

$$\hat{y} = \min\{i \in \{1, ..., K-1\},\ H(x) > \beta_i\}$$

That is, all alternatives $\mathbf{x}$ from class i are such that $t_i < H(x) \leq t_i + 1$. We recall that, as the output of a U-DAG-CI or UHCI, we have $H(\mathbf{x}) \in [0, 1]$, ensuring that all alternatives be able to be classified this way.

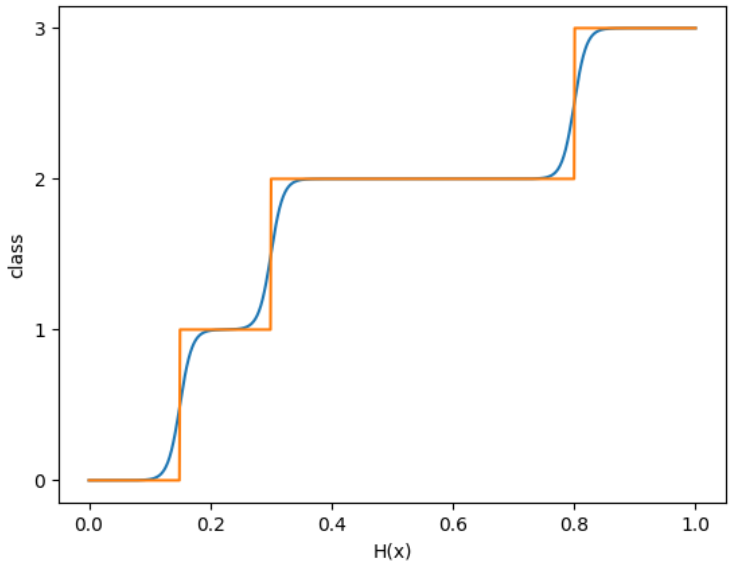


Figure 8.6: Class prediction (4 classes), given $H(x)$: sum of 3 step functions with means 3 sigmoids with respective thresholds 0.18, 0.29, 0.8, compared to a sum of 3 logistic sigmoids centered in 0.18, 0.29 and 0.8

8.5.4 Pairwise Preference Learning

Example 34. *We assume the houses in table 8.5, defined on the criteria from example 1. The alternatives are numbered from* 1 *to* 5 *according to their row in the table. It is analogous to bipartite object ranking, such as presented in Section 3.8.1.*

Then, we write $i \succ j$ *(resp.* $i \sim j$*) the information that the* i*th alternative is preferred to (resp. equally satisfying as) the* j*th alternative. Then, we will have a* DS*, of the following form:*

Surface m^2	Garden m^2	Garage (yes/no)	Road km	Transp. km	Downtown km	Price €
50	100	No	0.1	0.	0.	400,000
110	150	Yes	0.5	3.	4.	500,000
150	0	No	1.	0.5	0.5	450,000
150	30	No	0.1	5.	3.	300,000
500	1000	Yes	5.	5.	10.	1,500,000

Table 8.5: Alternatives 1 to 5 for the pairwise learning setting

- $1 \succ 2$ *(alternative* 1 *is preferred to alternative* 2*)*
- $1 \succ 4$
- $1 \succ 5$
- $3 \succ 1$
- $3 \succ 2$
- $3 \succ 4$
- $3 \succ 5$
- $5 \succ 2$
- $5 \succ 4$
- $2 \sim 4$ *(the alternatives are equivalently appreciated)*

That is, a set of pairs of preference and equivalence.

We wish to learn a UHCI or U-DAG-CI H which reflects those preferences. That is, we wish that, given a pair of alternatives $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ such that if $\mathbf{x}^{(1)} \succ \mathbf{x}^{(2)}$, we have $H(\mathbf{x}^{(1)}) > H(\mathbf{x}^{(2)})$. In the same way, we wish that, if $\mathbf{x}^{(1)} \sim \mathbf{x}^{(2)}$, then $H(\mathbf{x}^{(1)}) = H(\mathbf{x}^{(2)})$. Thus, H can still be interpreted as a score. Our dataset will have the form:

$$\text{DS} = \left\{ \left(\left((\mathbf{x}_{(j)}^{(1)}, \mathbf{x}_{(j)}^{(2)} \right), y^{(j)} \right), j \in \{1, ..., m\} \right\}$$

such that:

$$y^{(j)} = \begin{cases} 1 & \text{if } \mathbf{x}_{(j)}^{(1)} \succ \mathbf{x}_{(j)}^{(2)} \\ 0 & \text{if } \mathbf{x}_{(j)}^{(1)} \sim \mathbf{x}_{(j)}^{(2)} \\ -1 & \text{if } \mathbf{x}_{(j)}^{(1)} \prec \mathbf{x}_{(j)}^{(2)} \end{cases}$$

In order to learn from such data, we adopt a siamese approach [Bromley et al., 1993]. That is, we use not one, but two networks H_1 and H_2 with the same architecture and parameters (i.e. both are exact copies of each other and represent the same function). Given $\left(\left((\mathbf{x}_{(j)}^{(1)}, \mathbf{x}_{(j)}^{(2)} \right), y^{(j)} \right) \in \text{DS}$, the forward phase is such that H_1 is fed $\mathbf{x}_{(j)}^{(1)}$, and H_2 is fed $\mathbf{x}_{(j)}^{(2)}$. Their respective scores $S_1 := H_1 \left(\mathbf{x}_{(j)}^{(1)} \right)$ and $S_2 := H_2 \left(\mathbf{x}_{(j)}^{(2)} \right)$ are computed normally, by propagating through both networks parallelly. Then, the output of the siamese network is computed as:

$$2 * \sigma_{0,\gamma}(S_1 - S_2) - 1$$

With γ an hyperparameter that gives the steepness of the output sigmoid. Using this, we see that, indeed, if $S_1 = S_2$ (that is, if the model gives the same score to both alternatives), the output is 0. If $S_1 > S_2$, that is, if the model prefers S_1, the output goes towards 1, increasing with the difference $S_1 - S_2$. If $S_1 < S_2$, that is, if the model prefers S_2, the output goes towards -1, decreasing with the difference $S_1 - S_2$. This fits the $y^{(j)}$ as described above.

As we recall, they both represent the same function H (either a UHCI or a U-DAG-CI). Using the mean squared error as a loss function, we have:

$$\begin{aligned}
\text{Err(DS)} &= \frac{1}{m}\left(2 * \sigma_{0,\gamma}(S_1 - S_2) - 1 - y\right)^2 \\
&= \frac{1}{m}\left(2 * \sigma_{0,\gamma}\left(H_1\left(\mathbf{x}_{(j)}^{(1)}\right) - H_2\left(\mathbf{x}_{(j)}^{(2)}\right)\right) - 1 - y\right)^2 \\
&= \frac{1}{m}\left(2 * \sigma_{0,\gamma}\left(H\left(\mathbf{x}_{(j)}^{(1)}\right) - H\left(\mathbf{x}_{(j)}^{(2)}\right)\right) - 1 - y\right)^2
\end{aligned} \tag{8.2}$$

Now, the gradients can be computed on both networks H_1 and H_2. Indeed, assuming a parameter θ in H. Writing $O := S1 - S2$, we have:

$$\begin{aligned}
\frac{\partial \text{Err}(DS)}{\partial \theta} &= \frac{\partial \text{Err}(DS)}{\partial O} \cdot \frac{\partial O}{\partial \theta} \\
&= \frac{\partial \text{Err}(DS)}{\partial O} \cdot \left(\frac{\partial S_1}{\partial \theta} - \frac{\partial S_2}{\partial \theta}\right)
\end{aligned} \tag{8.3}$$

Then, both sets of gradients are applied to H_1, whose parameters are updated. Finally, H_2 is replaced by a copy of the updated H_1, such that both network still represent the same functions again.

8.5.5 Training and Regularization

8.5.5.1 Training Algorithms

These networks can be trained with any gradient descent algorithm available. In practice, as will be described with the experimental results, we had good results with stochastic gradient descent and batch gradient descent with simple geometric learning-rate decay. In many cases, momentum helped speed up the training phase by a significant factor.

8.5.5.2 Regularizations

In order to limit the number of parameters, and prevent overfitting, it is common to use regularizations. These appendages to the loss function penalize undesired behaviours, in particular keeping a large number of parameters.

This is in particular important for parameters that do not hold an intrinsic meaning, and are redundant with others. In particular, let M any monotonic marginal utility module (which might be part of a larger module, such as bitonic or selector modules), as defined in Chapter 6. We recall that the function represented by M can be written as shown in (6.1):

$$u(x) = \sum_{k=1}^{h} r^k s^k(x) := \sum_{k=1}^{p} \frac{r^k}{1 + e^{-(\eta^k x - \beta^k)}}$$

Due to the high redundancy between all sigmoids in this sum, we introduce the following $L1$ regularization term to the loss function on the parameters of module M:

$$R_M = \lambda \sum_{k=1}^{h} |r^k| \tag{8.4}$$

where λ is a non-negative hyperparameter determining the importance of the regularization. The higher λ, the more the model will try to eliminate sigmoids by reducing their weight to 0.

In practice, we need one such term for all monotonic marginal utility module (once again, whether they are used on their own or as a component of a bigger utility module).

8.5.6 Convergence of the Learning Settings

The three learning settings described in sections 8.5.2, 8.5.3 and 8.5.4 are not mutually exclusive. Indeed, in the three cases, the model will learn a UHCI or U-DAG-CI scoring model, with the given architecture, marginal utilities' types and aggregators' types.

Thus, a model trained in either types can be used as an inference models for either predicting a score (regression setting) or a preference (pairwise preference setting). Regarding the ordinal regression setting, the issue would be that a model trained in another setting would lack the information about the thresholds t_i that separate the classes. In any case, the scoring model might remain an informative tool for further inference on new data points. Its aggregators and marginal utilities can also be analyzed in order to obtain an interpretation of the data.

This also allows the model to be trained in hybrid data, with data points coming from either of the three settings, all within the same training dataset.

Example 35. *Below, we represent the alternatives from examples 31 as vectors of dimension 7 (one for each criterion). For instance, the first alternative of Table 8.3 would be written as* $(50, 100, \mathit{No}, 0.1, 0., 0., 400000)$. *For regression and classification, we write a labeled datapoint as a pair* $(\mathbf{x}, y)$ *with* $\mathbf{x}$ *the alternative and* y *the label.*

A mixed dataset, taking points into all three settings, would look like:

$$\begin{aligned}\mathrm{DS} = \{&((50, 100, \mathit{No}, 0.1, 0., 0., 400000), 0.4);\\ &((110, 150, \mathit{Yes}, 0.5, 3., 4., 500000), 0.1);\\ &((150, 0, \mathit{No}, 1., 0.5, 0.5, 450000), \mathit{Good});\\ &((150, 30, \mathit{No}, 0.1, 5., 3., 300000), \mathit{Bad});\\ &(50, 100, \mathit{No}, 0.1, 0., 0., 400000) \succ (500, 1000, \mathit{Yes}, 5., 5., 10., 1500000)\}\end{aligned}$$

The two first points are from the regression setting, the two next ones are from an ordinal regression setting, and the last one is a preference between two alternatives.

Of course, training on such a data set might prove tricky. First of all, there would be 3 learning rates to tune, rather than a single one. Secondly, as the data likely comes from different sources, it might be harder to ensure that the dataset is coherent (that is, that the optimal underlying decision model is similar for all sources). Nonetheless, if it is the case, mixing those settings allows to train on a larger dataset, in cases where data is hard to come by.

8.6 Conclusion and Remarks

We have seen in this part the heart of this thesis's contribution, namely the ability to learn a valid UHCI, or U-DAG-CI, end-to-end, from data of several types. These models are formally valid, and any possible model can be approximaged by a NEUR-HCI network. As a consequence, this ensures that thus trained models have all the desirable properties of their MCDA counterparts. In particular, they are easily interpretable, and can thus be easily verified by a field expert after training. They can also be analyzed at runtime by a DM, allowing their use in a safety-critical context. One last advantage is that, once trained, the neuronal structure of the model can be discarded, and the function represented can be written as a single mathematical formula, making it lightweight, quick to run,

and especially easy to embed on a dedicated frugal architecture, such as a FPGA board.

Moreover, the CI is a 1-Lipschitz-continuous function; as a consequence, so are HCIs and DAG-CIs. Assuming we can bound, by hand, the derivative of the marginal utilities to a given value γ, we can then obtain a γ-Lipschitz function. This can be done through clipping on the precision parameters of the logistic sigmoids involved in the model. Combined with the limited dimmensionnality of MCDA problems, this makes this model robust by-design to adversarial examples, even during training.

The monotonicity conditions also play a role in robustness. Indeed, if the DM can ensure that the model be valid on a given alternative $\mathbf{x}$, then the whole area of alternatives which $\mathbf{x}$ Pareto-dominates can only be considered at most as good as $\mathbf{x}$. If an expert can thus validate some critical outputs such as "unsatisfying", this thus allows to guarantee formally that the model will consider this region of the space as "unsatisfying", leading to the avoidance of errors on said region, regardless of any adversarial manipulation which remains in the said region. We now present some experimental results to validate our approach, both on real and artificial data. We show that the models presented in this part perform well, and are quite robust to noise [Bresson et al., 2020a]; this, together with the identifiability result presented in Part III, allows to expect some interesting stability in trained models' behaviours.

Part V

Experimentations and Empirical Validation

CHAPTER 9

EMPIRICAL VALIDATION OF NEUR-HCI

Contents

9.1 Experimental Setting and Objectives

9.1.1 General Considerations

In this chapter, we work on evaluating key aspects of the relevance of NEUR-HCI. We wish to evaluate, in particular, its advantages and drawbacks, in particular for using in safety-critical contexts.

In the first part of this chapter, we work in an MCDA-like setting. We assume that we know, from expert knowledge, a hierarchy fitting that of the underlying processes that generated the data. We show that a NEUR-HCI network, built with this information, is able to learn, very accurately, a model that fits the generated data well. Moreover, we show that the model is able to recover the real parameters of the generating model from this data only. The algorithms used for training are either batch gradient descent or stochastic gradient descent, with varying levels of momentum depending on the setting and hyperparameter tuning.

In the second part, we assume that the hierarchy and additivity is unknown. We thus study how well models with unknown hierarchies can be approximated by DAG-CI and flat models.

The main focuses of our evaluations:

- Performance: how well a NEUR-HCI model is able to fit data, and to learn a model with a low generalization error
- Stability/Robustness: how noise and random variations in data impact the performance and the learned model

Performance is obviously relevant to evaluate how well the model would perform on yet unseen data. It is the primary way of evaluating a machine learning model. Indeed, a model which performs poorly on new data is virtually useless for real life applications.

On the other hand, stability and robustness are necessary for interpretability and trustability. Indeed, if the same data can make models learn very diverse sets of parameters, there will be some conflicting interpretations, which, in turn, make the model impossible to interpret. While we showed the unicity of the model in Part III, this result is only theoretical, and does not provide any guarantees as to the behaviour of the model when training on noisy data, or on finite datasets. We

show, nonetheless, that empirical stability allows to reinforce this result, and hints at a stronger formal result for robustness.

9.1.2 How to Read a Figure ?

In the different settings in which we experiment, it is necessary to train several models to have a realistic evaluation of how well NEUR-HCI performs in this setting. Indeed, if we were to train only a single model in each setting, we could be good or bad "by chance", especially if the training is unstable. After training m models in the same context, we thus have m different values for a given parameter of the model, as each model has its own set of parameters. It is important to note that, unless stated otherwise in very specific cases, each model is trained on a close, but different dataset; for randomly generated points, this means that new points are drawn for each model. For real data, this means that we draw new random points in the dataset for each model. Some added variability comes from the random initialization of each model, which starts with random parameters (otherwise, models would likely all converge to the same basin of attraction, even if there are other local minima in the parameter space).

Stability would be ensured if, for each parameter, its m values across all models are close. As a consequence, we use boxplots to display the spread of a given parameter. An example of such a graph is given in Figure 9.1. This figure is read as follows:

- there is one sub-plot for each aggregator. In the case of a flat model, there is a single plot. In this case, we follow the hierarchy from Figure 2.2; there are thus 4 aggregators, with respective dimensions 3, 2, 2 and 3, and thus respectively 6, 3, 3 and 6 parameters in a 2-additive setting

- focusing on the first sub-plot (top left), we see 6 boxes (one for each parameter). Each box corresponds to a parameter. We chose to represent the interaction indices (see Section 2.5.5). There is thus one parameter $I(S)$ for each subset S of input of the given aggregator. They are ordered by size, then, for all subsets of equal size by lexicographic order. That is, given an aggregator of dimension 3, which aggregates criteria 1, 2 and 3, the parameters will be ordered as:

  1. $I(\{1\})$
  2. $I(\{2\})$
  3. $I(\{3\})$
  4. $I(\{1, 2\})$

5. $I(\{1,3\})$
6. $I(\{2,3\})$
7. $I(\{1,2,3\})$

- note that, for 2 additive aggregators, we thus only display the parameters $I(S)$ with $|S| \leq 2$. Indeed, all of the others are, by design, set to 0 as the learned measure is 2-additive. In our example, we thus do not display $I(\{1,2,3\})$, as it would be set to 0 for all models
- for an aggregator with k inputs, the k first parameters are thus the Shapley values of each criterion
- in all plots, the (red) stars show the ground-truth value, i.e. the value of the given parameter for the model that generated the data

The boxplots are to be read as such:

- the orange line is the median of the data points
- the lower and upper box's limits are respectively the quartiles $Q1$ and $Q3$
- let IQR be the interquartile range ($Q3$-$Q1$). The upper whisker will extend to last datum less than $Q3$+1.5*IQR. Similarly, the lower whisker will extend to the first datum greater than $Q1 - 1.5$IQR [1]
- the circles are the outliers (i.e. any point that is not in the range defined by the whiskers)

We thus expect for our boxplots to be around the star, and as flat as possible. For instance, we see that, in the top-right subfigure of Figure 9.1, the two first parameters are well learned with little variance, while the third ($I(\{1,2\})$ has high bias (the learned values are away from the ground truth) and medium variance.

9.1.3 Generator Models

We work in this section often with artificial data. That is we, construct a so-called *ground-truth* model Q. Then, we randomly draw some data points $(x^{(1)}, ..., x^{(m)})$ in X, and compute their expected labels as $y^{(i)} = \mathrm{Q}(x^{(i)})$. In the case of pairwise preference data, we randomly draw a, b two alternatives in X, and add the pair (a, b) to the data if and only if $\mathrm{Q}(a) > \mathrm{Q}(b) + \delta$, with δ the threshold for strict

[1] https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.boxplot.html

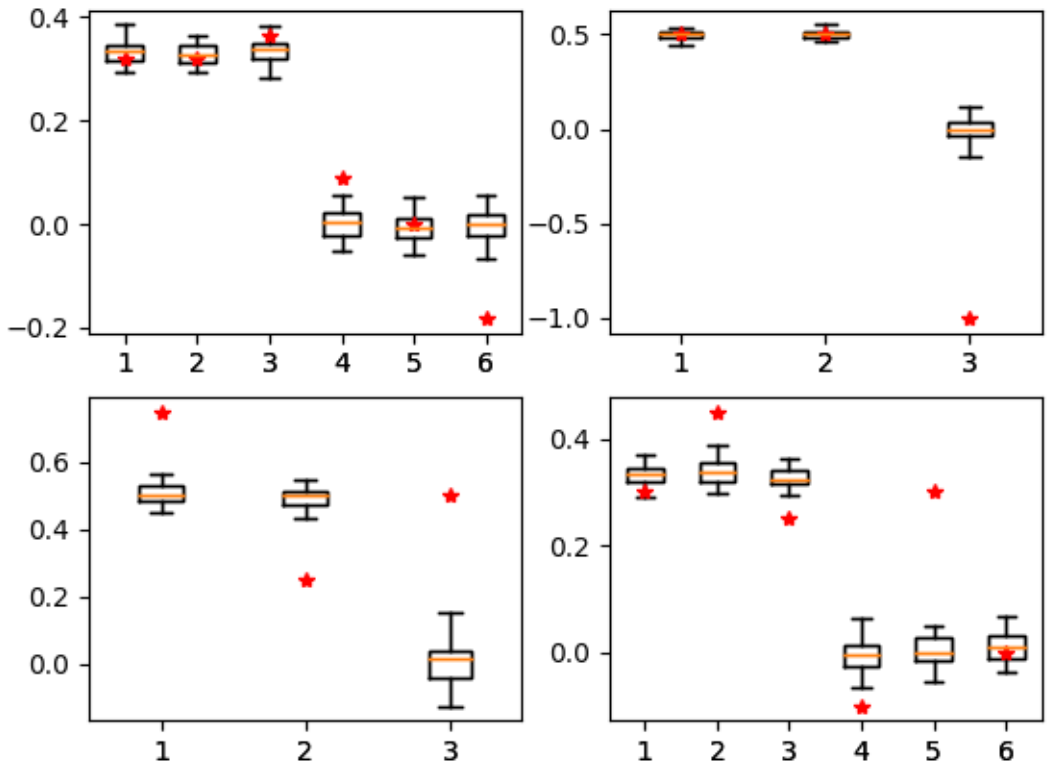


Figure 9.1: Example of Figures. Each of the 4 sub-plots represents a given aggregator. Note that this does not represent any real experiment, and is only there to present the structure.

preference. In our case, we used $\delta = 0.1$, as this denotes a significant preference while remaining reasonably small.

We can thus control many aspects of the data generation, including its volume, noisiness, and the complexity of the underlying phenomena.

We present in this section two standard models that we used in all settings for ability to compare. They have 2-additive aggregators, and were used to generate data in dimension 7. We refer to them below as *standard tree model* and *standard flat model*, respectively. When their use is not specified, then a model was randomly generate for the specific test.

9.1.3.1 Standard Tree Model

For generating artificial data, we build models, the parameters of which we know. Nonetheless, for some tests, we built by hand a custom model, in dimension 7, so that we have some consistency among the different settings. This model has the same hierarchy as Figure 2.2, which we recall below in Figure 9.2. Its aggregators have the following expressions:

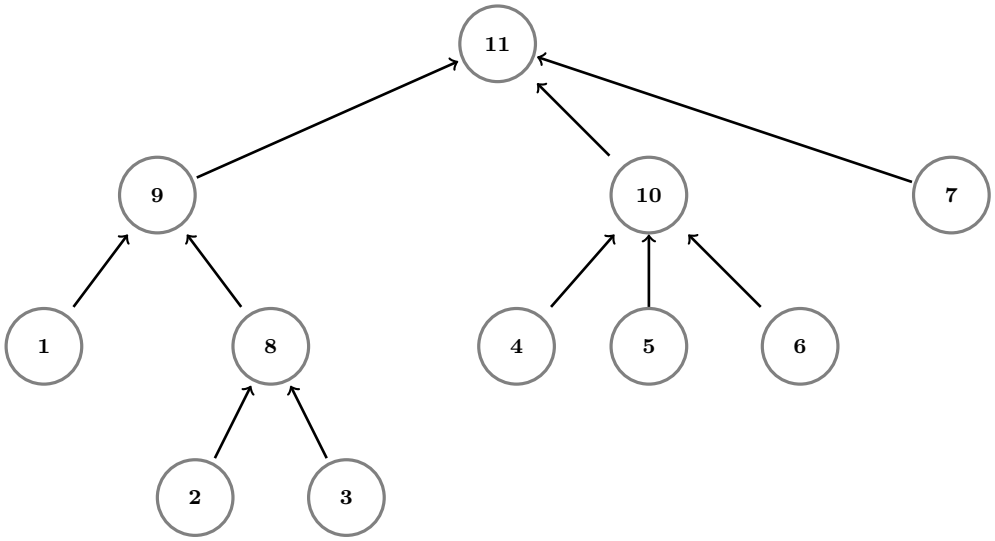


Figure 9.2: Hierarchy of criteria described in Example 1.

- $x_i = a_i$ for $i \in \{1, ..., 7\}$
- $a_8 = \frac{a_2 + \min(a_2, a_3)}{2}$
- $a_9 = \max(a_1, a_8)$
- $a_{10} = 0.1a_4 + 0.4a_5 + 0.1a_6 + 0.3\min(a_4, a_6) + 0.1\max(a_5, a_6)$
- $a_{11} = \frac{3a_7 + 2a_9 + 3a_{10} + \min(a_7, a_9) + 2\max(a_9, a_{10})}{11}$

Notice that aggregator 9 is a single max, which violates the assumption for identifiability. Nonetheless, we assume in this setting that the hierarchy be fixed; as a consequence, this is not an issue, since the model with a fixed hierarchy remains identifiable even in this case by Theorem 1 in Chapter 4.

We illustrate in Figure 9.3 what each point corresponds to in the case of the standard tree model. The node indices are those of figure 9.2. Note that these are the local importance of each node w.r.t. its parent (Shapley values and interaction

indices of each individual CI). We also give the global Winter value(as defined in Section 2.7.3) of each leaf, from 1 to 7 respectively (rounded at 4 decimals): (0.159, 0.1193, 0.0397, 0.109, 0.1636, 0.0909, 0.3181). We notice that criterion 7 is thus the most important (~ 0.32), while crieterion 3 is the least important (~ 0.04).

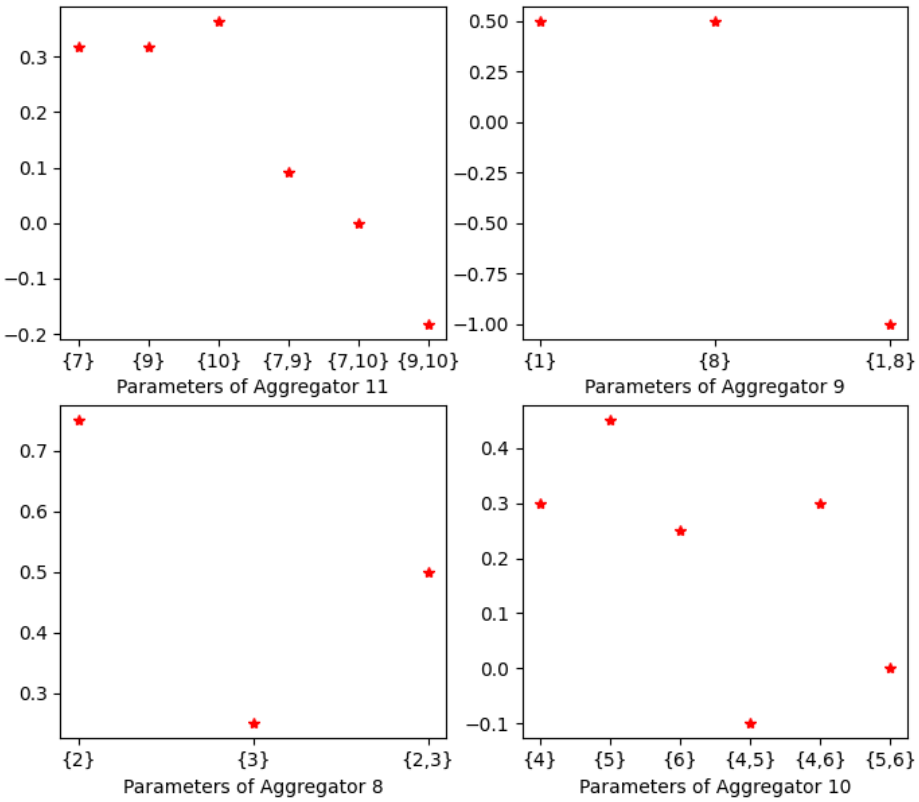


Figure 9.3: Ground truth values of the interaction indices of each aggregator for the standard tree model. Note that the x-axis is labeled by the set to which the parameters correspond. If a set is a singleton, then its interaction index is also its Shapley value. Note that these indicators are computed at a local aggregator level, they do not reflect the general importance of a criterion in the model (see the Winter values given in Section 9.1.3.1 to know the global importance of all criteria).

9.1.3.2 Standard Flat Model

The standard flat model is a single, 2-additive CI in dimension 7. It can be written as:

$$\begin{aligned} C_\mu(x) =& 0.1x_1 + 0.2x_2 + 0.05x_4 + 0.05x_5 + 0.2x_7 + \\ & 0.05\min(x_1, x_3) + 0.05\min(x_3, x_5) + 0.1\min(x_4, x_6) + 0.2\max(x_2, x_4) \end{aligned} \tag{9.1}$$

The Shapley values of each criterion on this model are, from criterion 1 to 7 (rounded at 4 decimals): (0.1249, 0.2999, 0.0499, 0.1999, 0.0749, 0.0499, 0.1999). Note that due to this model being flat (single CI), its global Winter values are the same as its Shapley values.

9.2 Training with a Fixed Hierarchy

9.2.1 General Settings

In this setting, we generate artificial data from a known model Q with hierarchy $\mathcal{T}$. Then, we build NEUR-HCI network with the same hierarchy $\mathcal{T}$ and train them on the given data. We show that the trained models not only achieve nice performance on the data, but also that they are able to recover the true underlying parameters of the generator.

In the same way, we allowed the trained models to learn marginal utilities if and only if the generating model did use some to generate its data.

9.2.2 Consistent Data, Large Sample Limit

We assume, at the moment, that the data is not noisy. That is, for all $(x, y) \in \mathrm{DS}$, $y = Q(x)$. This means that, by Theorem 12, we can approximate Q as closely as possible with the right parameters. We show that, in practice, we do converge to the right parameters.

In this section, we generated a large number of data points. In all cases, both training and testing error converged to values close to 0, we usually stopped fine-tuning the hyperparameters once we reached a mean squared testing error (on a testing dataset of $10,000$ points) of 10^{-7}.

Figure 9.4 shows the distribution of parameters of 50 models, trained on data generated by the standard tree models. We recall that the models we trained have the same hierarchy as the generating one. We thus see 4 subplots, corresponding to the parameters of aggregators 11, 8, 9 and 10 (in that order from left to right, top

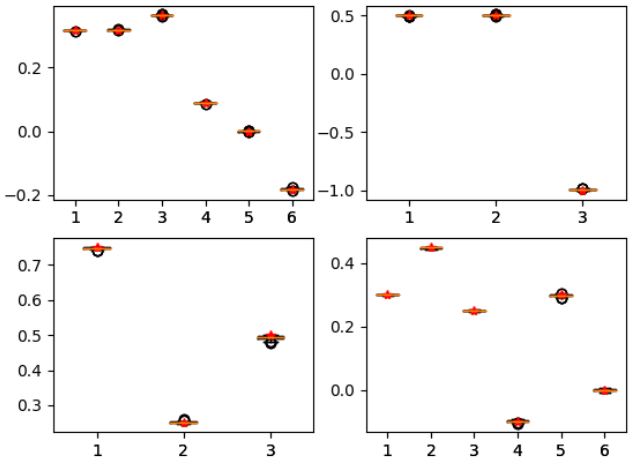


Figure 9.4: Distribution of the parameters of the aggregators of a UHCI with C2 aggregators, 10,000 training examples generated by a UHCI with 2-additive aggregators. 50 models trained in regression setting.

to bottom). We can clearly see that the parameters for all aggregators are learned with very little bias and variance. Indeed, the boxes are so flat we cannot see any significant spread, and they are centered on the values of the same parameters for the generator model (the red stars). Even the outliers (circles) are very close to the real values.

Figure 9.5 shows a similar ability to converge to the right values, but on a flat model of the same dimension, with thus more parameters. We also plot in Figure 9.6 the marginal utilities learned of all models on the 7 dimensions. We see that they are all superimposed over the real ones, which were used to generate the data.

We see that using a model with higher additivity is not a problem. For instance, Figure 9.7 (learning 2-additive data with a general aggregator) shows that the parameters corresponding to sets of cardinality greater than 2 have converged to 0, i.e. their real values considering that the generator is 2-additive.

Similar pictures for learning models with the same hierarchy as the generator in the large sample limit can be found in Appendix A.1. In all cases, we notice that the result is very stable, always centered around the right parameters. In other words, given enough non-noisy data, and assuming that the generative model is a UHCI, NEUR-HCI is able to recover the true parameters of the underlying model.

This last fact suggests a stronger result than the theoretical identifiability; indeed, it hints at the fact that the unique theoretical solution is also the only local minimum of the practical loss function, given a fixed hierarchy. This, in

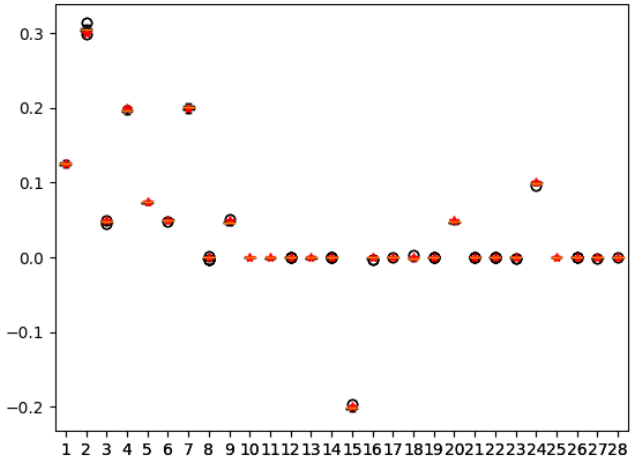


Figure 9.5: Distribution of the parameters of the aggregators of a flat UHCI with a 2-additive aggregator, 10,000 training examples generated by the flat standard model. 50 models trained in regression setting.

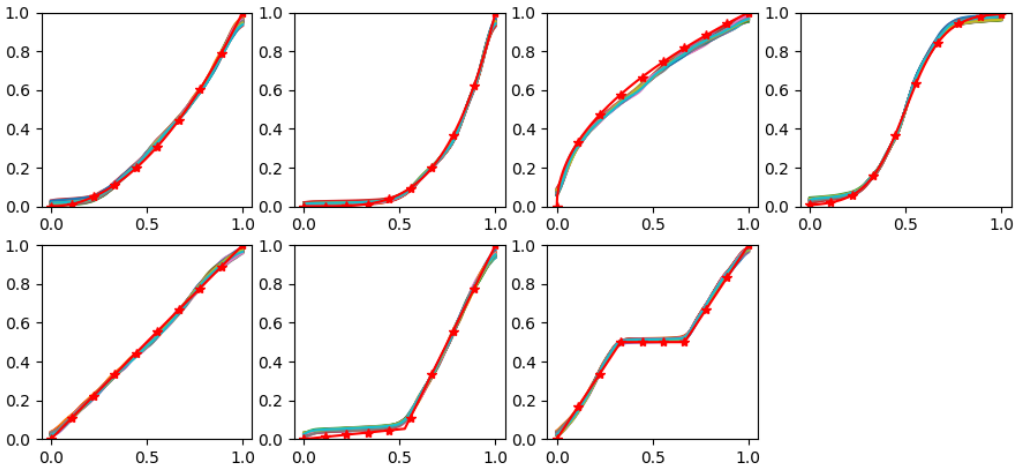


Figure 9.6: Plotting the marginal utilities of 50 models trained in regression setting, 10,000 training examples generated by a UHCI, and trained on a UHCI with similar hierarchy. The ground-truth marginal utilities are marked with crosses.

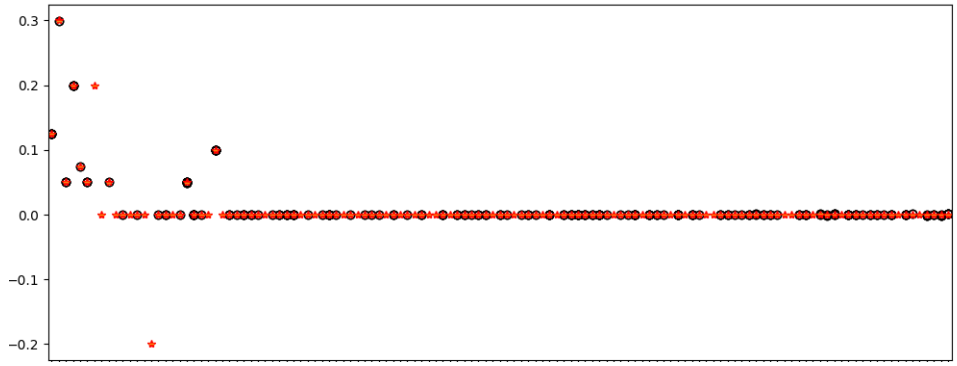


Figure 9.7: Distribution of the parameters of the aggregators of a CI with CG aggregators, 10,000 training examples generated by an HCI with 2-additive aggregators. 50 models trained in regression setting.

turn, encourages theoretical work on proving the existence of a unique basin of attraction in the parameter set, reinforcing the identifiability result.

9.2.3 Selectors

The tests below were made on artificial data, to test the ability of the new modules to learn models which are known perfectly. In order to train the selectors, we create a UHCI model Q (with known parameters). Note that the model is not a NEUR-HCI network, as the marginal utilities are explicitly defined as mathematical functions; this means that they can only be approached by a Neur-HCI with a finite number of sigmoids (we use 100 in these experiments).

Finally, we train 100 Neur-HCI models with the most general selectors in place of utility modules (M_1 is single-peaked, M_2 is single-valleyed). Note that a new dataset is drawn for each model. These selectors can choose between the four types of marginal utilities, both monotonic and bitonic ones. We then compare the type of the learned utilities to their ground-truth type, The results are presented in the confusion matrix below. Each line and column correspond to a type of marginal utilities, with: "ND": non-decreasing; "NI": non-increasing; "SP": single-peaked and "SV": single-valleyed. For instance, the number in square from row "ND" and column "SV" shows how many times the model learned a single-valley function when the ground truth was "non-decreasing".

Real Type	ND	NI	SP	SV
ND	0.993	0.0	0.007	0.0
NI	0.0	1.0	0.0	0.0
SP	0.0	0.01	0.99	0.0
SV	0.0	0.0	0.0	1.0

Table 9.1: Confusion matrix, 6 criteria, 1000 generated examples

We see that there are very few errors here, showing NEUR-HCI's ability to learn the right type of utilities. This is coherent with the identifiability of a UHCI fitted with utilities of these types [Labreuche et al., 2016]. Moreover, there is never any confusion between "opposite" types of utilities: a non-increasing one (resp. single-peaked) is never identified as a non-decreasing (resp. single-valleyed) one, and vice versa.

The remaining error nonetheles hints at the fact that there is high sensitivity in the criteria used for re-characterizing a given bitonic marginal utility into a monotonic one. There might thus be some work needed in order to robustify this process, such that those models can work nicely even in noisy settings.

9.2.4 Bitonic utilities

We trained 100 models (4-dimensional, with one marginal utility of each type) on 1000 randomly generated examples (different for each models). Figure 9.8 shows the graph of all of the learned marginal utilities (100 per graph), with the ground-truth values as stars. As we see, there is little variation between the learned functions, which fit the ground-truth well. The most sensitive areas seem to be the threshold in the bitonic case, as the maximal variance is observed there.

Note that the parameters of the aggregators also showed very little variance, and fitted the ground truth model well (the maximal error between a learned weight and its true value was 0.02, the average error on all weights and all models was 0.013). This is illustrated in Figure 9.9; note that we display the Möbius values (see Section 2.11) here due to using an older version of the software being used, the information contained is the same as when displaying interaction indices.

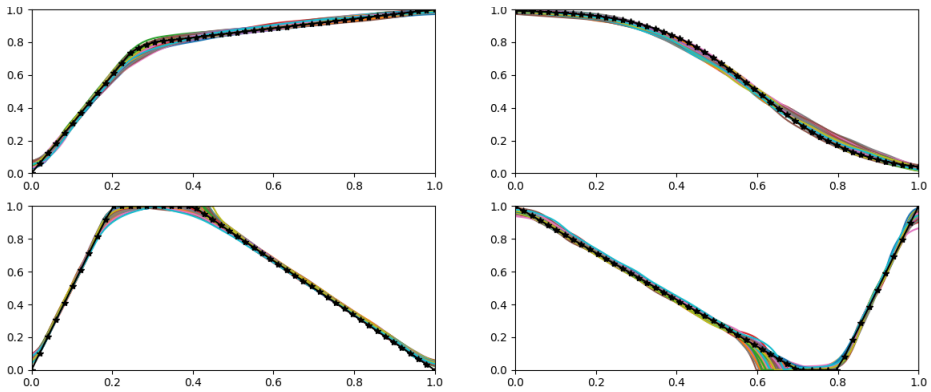


Figure 9.8: The four types of marginal utilities, learned by 100 models on 1000 artificial, noiseless examples. Black stars denote the real values.

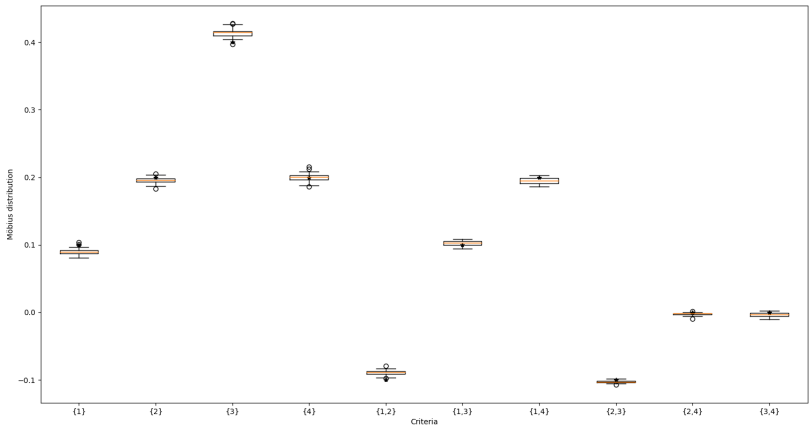


Figure 9.9: Distribution of the CI parameters (Möbius values) of the model from Figure 9.8

9.2.4.1 Discussion

We saw above that NEUR-HCI is able to recognize the direction of marginal utilities (among the four defined behaviours) in most cases, with artificial data. Moreover, it yields high stability both in the parameters of the aggregators and the values of the learned marginal utility functions. In all settings, the trained models converged to the right parameters. In particular, we see in most figures that the boxes are very flat, and centered on the ground-truth, which is exactly what we seek (very low error, very low variance).

This opens an opportunity in using NEUR-HCI as an analysis tool; in a case where a user would wish to know whether the type of the relationship between several criteria and an output (or a label), a NEUR-HCI can be trained on a

dataset and show which values increase (or decrease) the output. In domains where the user can affect the value of the criteria for building alternatives (in a design setting, for instance), this can in turn be used to optimize the values of the criteria w.r.t. a given metric; by increasing (resp. decreasing) the criteria with an increasing (resp. decreasing) utility, or by aiming towards the peak (resp. valley) when the utility is bitonic. By doing this, a user could build better alternatives or improve existing ones.

This could also find application in black box optimization model, where parameters can be meaningless (and thus not have an obvious relationship with the output), in order to find whether several parameters tend to affect the output positively or not, and thus indicating directions or regions to explore, using the parameters of the aggregator as indicator of their relative importances.

Indeed, in some technical cases, we might have datasets which reflects certain phenomena that experts might be unaware, or unsure, of. As a consequence, training a NEUR-HCI model on such data might allow, by studying its parameters, to determine:

- which attributes contribute positively/negatively w.r.t. the output
- which attributes contribute the most/the least
- which attributes are in synergy/redundancy/independance
- which values on a given attribute increases greatly or not the output

Example 36. *Assume the realtor from Example 1 has a list of all houses sold in the past years, along with a satisfaction grade given by the customers. By training a model to predict this satisfaction, then looking at the model's Shapley values, interaction indices and marginal utilities, the realtor can determine which parameters are the most important for the customers.*

Assuming that the learned parameters show that having a garage is highly redundant with having a garden, but that the Shapley value of the garden parameter is larger than that of the garage, the realtor knows that they had rather build houses with gardens than with both a garage and a garden. This way, they maximize the satisfaction while reducing the costs.

This stability is also encouraging, as it seems to indicate that there is a single basin of attraction in the loss function, reinforcing empirically the identifiability result presented in Part III. Moreover, the learned values are always the same as the real ones, i.e. this basin of attraction is around the ground-truth model. In practice, this designates the real model as the single local minimum of the function. This is highly relevant as bad local minima are a hindrance for using gradient-based learning.

Moreover, we have seen that, in safety-critical settings, it is necessary for an expert to be able to validate the model in order to trust it. The fact that training a model on data that differs slightly yields a similar result allows to expect the model to be able to cope with the randomness inherent to the data. Moreover, the unicity of the learned model means that an expert only has to validate a single parameterization in order to approve or reject the model.

All in all, these empirical results confirm the interest for a theoretical study of the shape of the loss function when learning the parameters of an HCI. Such results could confirm the ease of learning such models from data.

Nonetheless, the real world applications are usually accompanied by noisy and small datasets. Our next experiments thus focuses on such settings.

9.2.5 Smaller and Noisy Datasets

In this setting, we keep the assumption that the hierarchy is known. Nonetheless, the datasets used to train our models will be relatively smaller and noisy. Given a non-noisy example (x, y) and a noise level ϵ, we generate an ϵ-noisy example $(x, \tilde{y})$ with the following processes, depending on the setting:

- Regression: $\tilde{y} = \mathrm{Q}(x) + \mathcal{N}(0, \epsilon)$
- Binary Classification: $\tilde{y} = y$ with probabilty $1 - \epsilon$, $1 - y$ otherwise
- Pairwise preferences: given a non-noisy example $x_1 \succ x_2$, we keep it intact with probability $1 - \epsilon$, and invert it as $x_2 \succ x_1$ otherwise

9.2.5.1 Performance

We present here the errors (computed on large, non-noisy datasets) yielded by models trained on smaller and noisy datasets, in the different settings. Each time, once again, we train models with the right hierarchy. Table 9.2 shows the mean and standard deviation of the mean squared error for the regression setting, 9.3 shows the misclassification error (proportion of misclassified examples) for the binary classification setting and 9.4 shows the mis-ranking error (number of erroneous preferences) for the pairwise learning-setting. The errors were computed on $10,000$ non-noisy examples, to see how well the trained model is close to the underlying source.

Noise	Tree	Flat
0.	$7.1 \times 10^{-7} \pm 4.4 \times 10^{-6}$	$2.5 \times 10^{-6} \pm 7.2 \times 10^{-7}$
0.05	$1.1 \times 10^{-4} \pm 4.6 \times 10^{-5}$	$1.7 \times 10^{-4} \pm 4.8 \times 10^{-5}$
0.1	$4.4 \times 10^{-4} \pm 1.6 \times 10^{-4}$	$6.3 \times 10^{-4} \pm 1.8 \times 10^{-4}$
0.2	$1.6 \times 10^{-3} \pm 7.6 \times 10^{-4}$	$1.7 \times 10^{-3} \pm 6.1 \times 10^{-4}$

Table 9.2: Mean Squared Error in regression setting, 300 examples generated by the standard Tree models (*Tree* column) and the standard Flat model (*Flat* column). The models learned have the right hierarchy. The error is computed on 10,000 noiseless data points.

Noise	Tree	Flat
0.	0.02947 ± 0.00592	0.02741 ± 0.00493
0.05	0.06322 ± 0.00747	0.04792 ± 0.00741
0.1	0.06708 ± 0.01019	0.05679 ± 0.01209
0.2	0.09848 ± 0.02276	0.08067 ± 0.01579

Table 9.3: Misclassification Error (proportion of points that were not assigned to the right class), binary classification setting. Training sets composed by 300 examples generated by the standard Tree models (*Tree* column) and the standard Flat model (*Flat* column). The models learned have the right hierarchy. The error is computed on 10,000 noiseless data points.

Noise	Tree	Flat
0.	$3.2 \times 10^{-5} \pm 0.00012$	$9.5 \times 10^{-5} \pm 0.00032$
0.1	0.00731 ± 0.00911	0.01843 ± 0.0257
0.2	0.02676 ± 0.02331	0.04047 ± 0.04261

Table 9.4: Mis-Ranking Error (proportion of inverted pairwise preferences), pairwise ranking setting, Training sets composed by 300 examples generated by the standard Tree models (*Tree* column) and the standard Flat model (*Flat* column). The models learned have the right hierarchy. The error is computed on 10,000 noiseless data points.

Note that, in all of these cases, when the noise was 0, the training error tended to 0 or was marginally close, a sign that the models were always able to fit the training data well.

In the regression setting, the testing error remains very low when there isn't any noise; it obviously increases with the noise, but always remains significantly inferior to the noise level (note that the reported error is squared). This is probably because regression data is very informative, giving a precise target to the model.

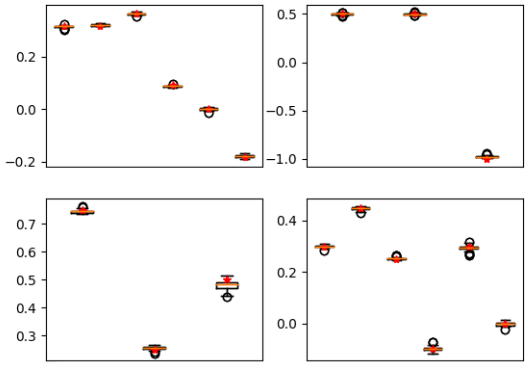


Figure 9.10: Distribution of the parameters of the aggregators of an HCI with C2 aggregators, 100 training examples generated by an HCI with 2-additive aggregators. 50 models trained in regression setting.

We see that the mis-ranking error in the pairwise-preference learning is quite good as well, always inferior to the noise level of the training data. While there is a significant difference between both columns, this does not mean that Tree models perform better, as each column was trained on different datasets from different models.

In the classification setting, we notice some level of error, even without noise. Note that the training error usually converged to values close to the noise level (as expected). In particular, the error was 0 (or marginally different) for non-noisy training sets. This indicates overfitting; i.e. the level of information contained in the dataset was too little for such flexible models; they were thus able to perfectly fit the training set, but have difficulty generalizing well. Moreover, it is to be noted that the data generation method used was a uniform sampling of the domain, rather than the exponential distributions that are assumed by the use of a logistic loss.

All in all, these examples show that NEUR-HCI models are able to train even on reasonably-sized and noisy datasets, and maintain good performance. Examples on real data can be found later in Section 9.4.

9.2.5.2 Stability

We present here how the model's parameters behave when trained on small random datasets. Other settings and aggregators can be found in Annex A.2. When not otherwise specified, we trained in the regression setting.

As we see, training with 100 datapoints is enough for a model to learn good approximations of the parameters of an HCI. The marginal utilities, on the other

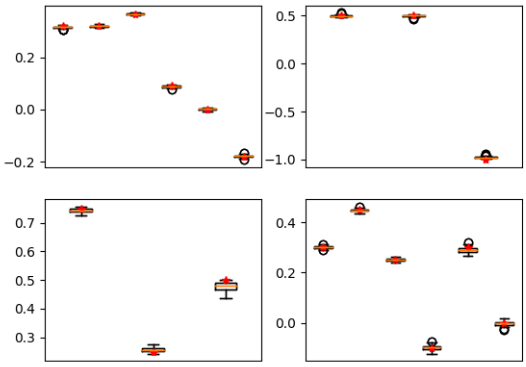


Figure 9.11: Distribution of the parameters of the aggregators of a UHCI with C2 aggregators, 100 training examples generated by a UHCI with 2-additive aggregators. 50 models trained in regression setting.

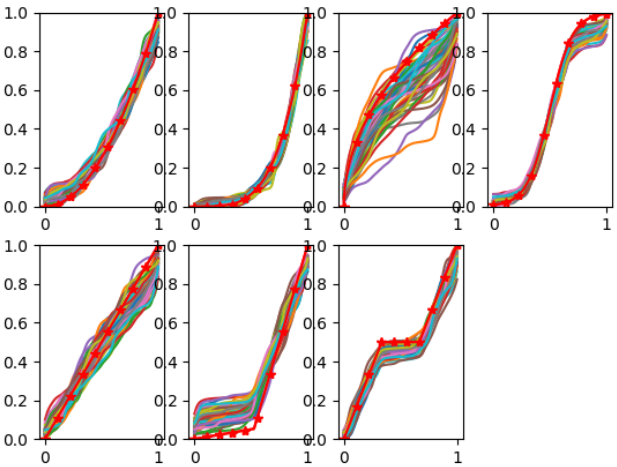


Figure 9.12: Plotting the marginal utilities associated with the model of Figure 9.11. Note that the criteria with the smallest importance (3 and 6, with respective global Winter values of 0.04 and 0.09) are also the one with the most variance.

Aggregator	Tree	Flat
C2	0.00018 ± 0.00012	0.00032 ± 0.00022
C3	0.00021 ± 0.00016	0.00041 ± 0.00025
CG	0.00062 ± 0.00041	0.00327 ± 0.00104

Table 9.5: Mean squared testing error of the model trained on 30 randomly selected points. The error was computed on a testing dataset with $10,000$ examples, the training labels were computed by the standard models.

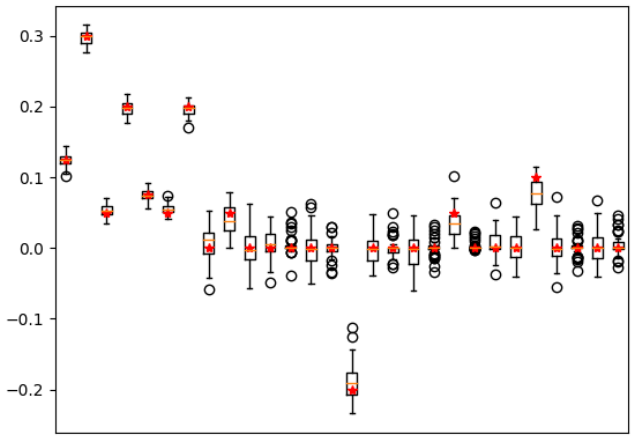


Figure 9.13: Learned parameters of 50 flat 2-additive UCI models trained on data generated by a flat 2-additive UCI. The training set was composed of 1000 examples, with a gaussian noise with standard deviation 0.05

hand, suffer a clear loss of quality; in particular, as we can expect, the attributes with the lowest influence on the model see their marginal utility more degraded. This is understandable, as an error on a criterion with small importance will yield to a small global error, which might be compensated several times by an error on a much important criterion, not allowing the model to correct the model perfectly. It is recurrent in the figures to thus notice that criterion 3 is the least-well learned each time, as it has a very low importance (0.04).

With 30 examples, the parameters learned are even more degraded, and the marginal utilities become imprecise at best. Table 9.5 shows the mean squared error of models trained on 30 random points, and tested on $10,000$ points from the same generator.

We illustrate here, as well as in Appendix A.3 the boxplots of the parameters of models trained with noisy datasets.

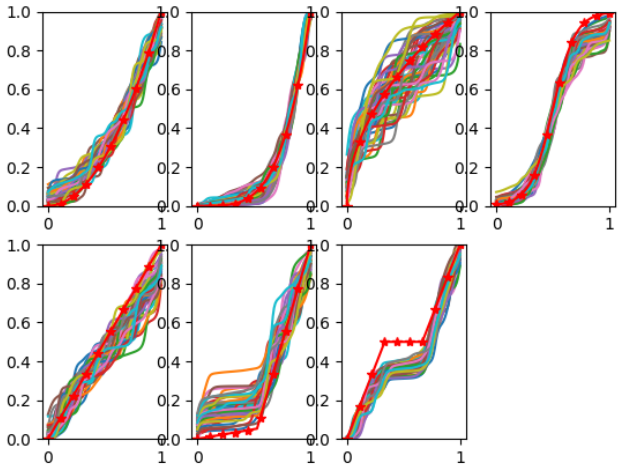


Figure 9.14: Marginal utilities for the model from Figure 9.13.

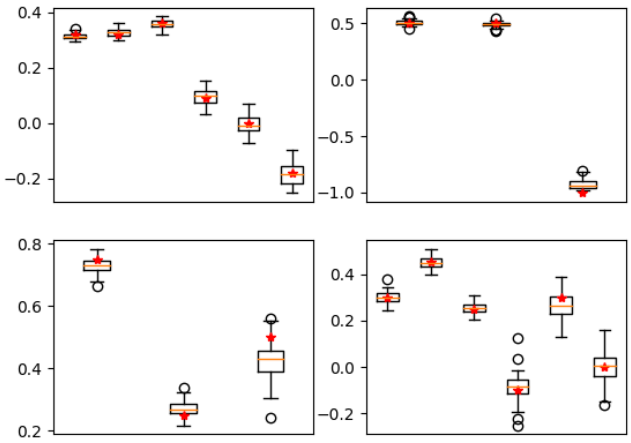


Figure 9.15: Learned parameters of 50 2-additive UHCI models trained on data generated by a 2-additive UHCI. The training set was composed of 1000 examples, with a gaussian noise with standard deviation 0.05

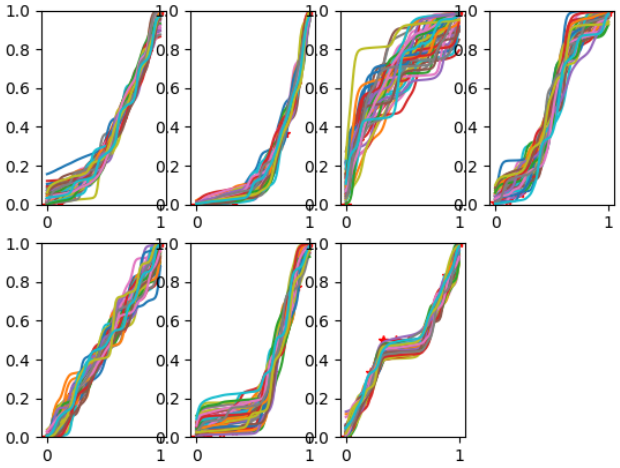


Figure 9.16: Marginal utilities for the model from Figure 9.15.

We notice an obvious degradation of the learned parameters, in particular the marginal utilities. Nonetheless, we see that the parameters of the flat model are submitted to much higher volatility/variance than those of the tree model. In particular, in all cases, the Shapley values of all aggregators (i.e. the first k parameters, with k the dimension of the said aggregator) are quite accurately obtained each time. Nonetheless, such variance might make it quite hard to interpret a trained model.

9.2.5.3 Corner Points - Binary Alternatives

Obviously, a small number of uniformly drawn values is not really informative; in particular, there might be lots of information shared between points (redundancy). As a consequence, and given the constrainedness of the model, we consider whether a limited set of carefully chosen points might suffice in eliciting the real model.

In this setting, we generate data points that come from the corners of the unit hypercube. That is, a given datapoint can be written as $(0_{-S}, 1_S)$ for a given $S \subseteq N$. These are also called binary alternatives.

We try then to learn an HCI model based on this data. Note that we do not consider marginal utilities, as such a dataset would hold no information about them (we assume to sample values only in the extrema of the interval domain of each criterion).

The idea is that, depending on the additivity of the model, a fixed number of such points are enough to fully determine a model through a combinatorial process,

as we showed in Chapter III. Each time, the label is computed by the standard model.

We study two cases:

- 30 points drawn randomly from the 128 corners
- all 128 corners of the unit hypercube

The results can be observed in Figures 9.17 to 9.18, with more settings in Appendix A.4.

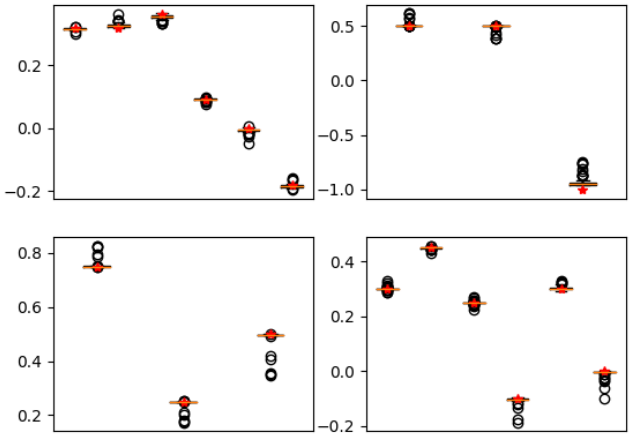


Figure 9.17: Distribution of the parameters of the aggregators of an HCI with C2 aggregators, trained on 30 randomly drawn corners of the unit hypercube (different for each trained model) , with the standard tree model as the ground truth.

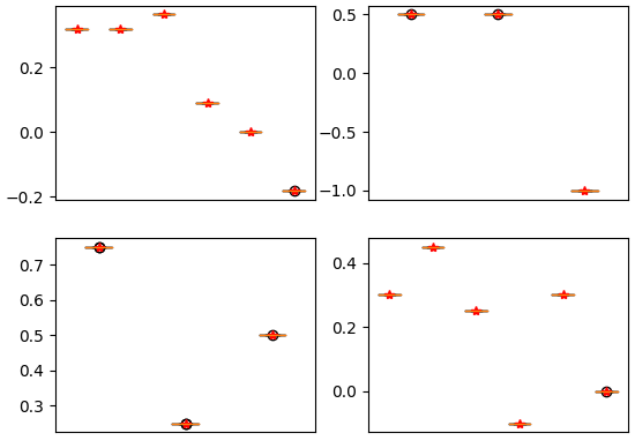


Figure 9.18: Distribution of the parameters of the aggregators of an HCI with C2 aggregators, trained on all 128 corners of the unit hypercube, with the standard tree model as the ground truth.

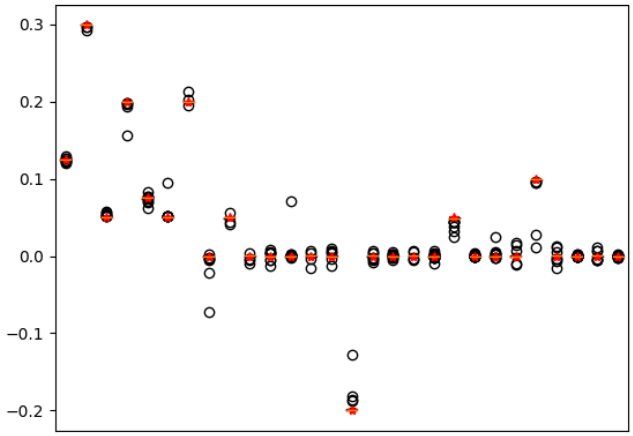


Figure 9.19: Distribution of the parameters of the aggregators of an HCI with C2 aggregators, trained on 30 randomly drawn corners of the unit hypercube (different for each trained model) , with the standard flat model as the ground truth.

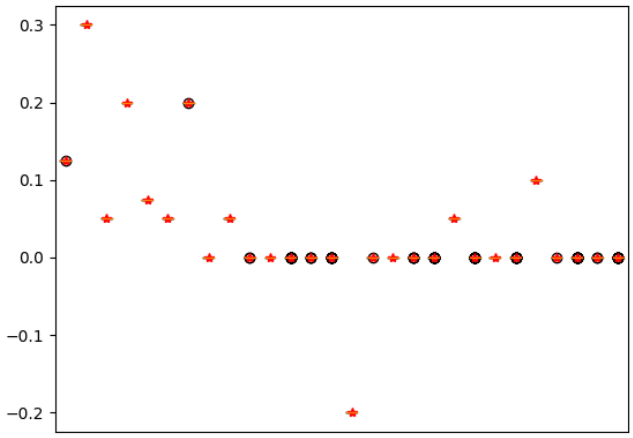


Figure 9.20: Distribution of the parameters of the aggregators of an HCI with C2 aggregators, trained on all 128 corners of the unit hypercube, with the standard flat model as the ground truth.

We see on these pictures that, when given all of the corners, all of the models, be it trees or flat, are able to recover the right parameters with little variance. On the other hand, when given only 30, there is still variance among the parameters. This is expected, as these points are not enough to actually define the model totally (that is, several models could fit the data perfectly); the models are thus highly vulnerable to the random drawing of the 30 points among 128. Table 9.6 shows the error in training trees and flat models with different aggregators with those 30 corners. We notice that, as the aggregators get more complex, the error increases. There is, nonetheless, a clearly lower error on tree models than on the flat one, as is expected as the number of parameters to elicit is lower.

Aggregator	Tree	Flat
C2	$4.4 \times 10^{-7} \pm 1.3 \times 10^{-6}$	$2.2 \times 10^{-5} \pm 9.7 \times 10^{-5}$
C3	$2.3 \times 10^{-5} \pm 0.00013$	0.00033 ± 0.00018
CG	0.00020 ± 0.00042	0.00052 ± 0.00194

Table 9.6: Mean squared testing error of the model trained on 30 randomly selected corners of the unit hypercube. The error was computed on a testing dataset with $10,000$ examples, the training labels were computed by the standard models.

In effect, the training error converged quickly to very low values (we stopped when we reached the 10^{-8} range) in all settings. In particular, the variance can be

observed on parameters that correspond to interactions among large coalitions of criteria. We see that, in terms of performance, all settings are better than on 30 points drawn randomly in the input space.

We also trained models on an even smaller dataset that fully defined the parameters of a given model (that is, a set of points that would be enough to elicit the model with combinatorial methods):

- all corners written as $(1_S,\ 0_{-S})$ with $1 \leq |S| \leq 2$ if the learned aggregators are 2-additive: 27 points in dimension 7

- all corners written as $(1_S,\ 0_{-S})$ with $1 \leq |S| \leq 3$ if the learned aggregators are 3-additive: 62 points in dimension 7

Notice that we exclude the all-zero vector, as, by design, the model would return 0 regardless of its parameters. In all cases, the models converged to the right parameters, as illustrated in Figures 9.21 to 9.24 and in Annex A.4.

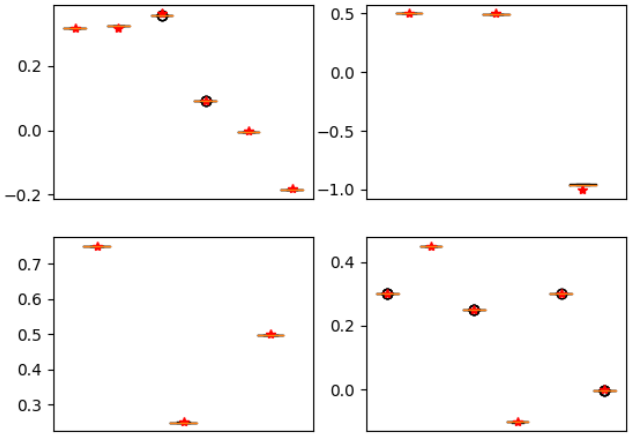


Figure 9.21: Distribution of the parameters of the aggregators of an HCI with C2 aggregators, trained on the 27 corners of the unit hypercube with at most 2 coordinates set to 1. The labels are computed by the standard tree model.

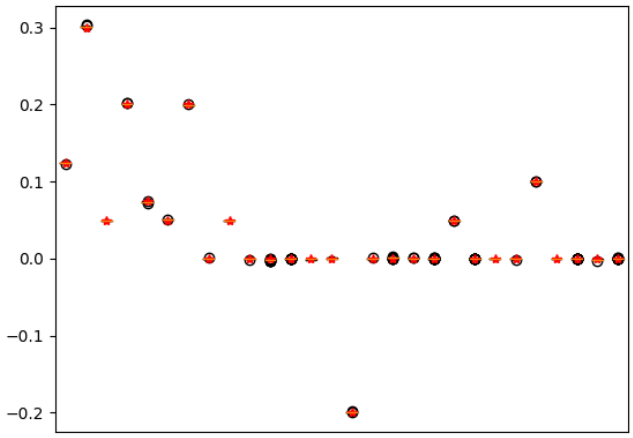


Figure 9.22: Distribution of the parameters of the aggregators of a CI with C2 aggregators, trained on the 27 corners of the unit hypercube with at most 2 coordinates set to 1. The labels are computed by the standard flat model.

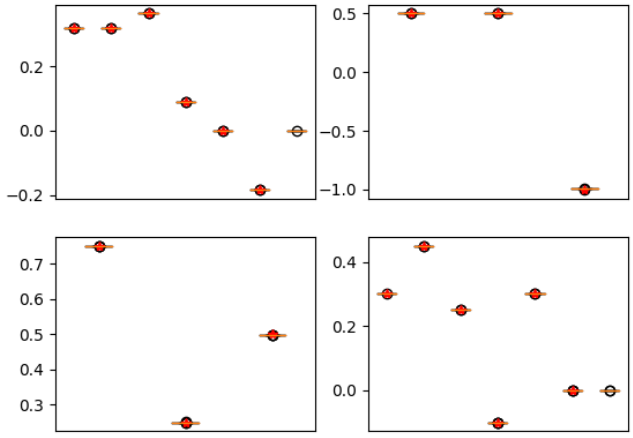


Figure 9.23: Distribution of the parameters of the aggregators of an HCI with C3 aggregators, trained on the 62 corners of the unit hypercube with at most 3 coordinates set to 1. The labels are computed by the standard tree model.

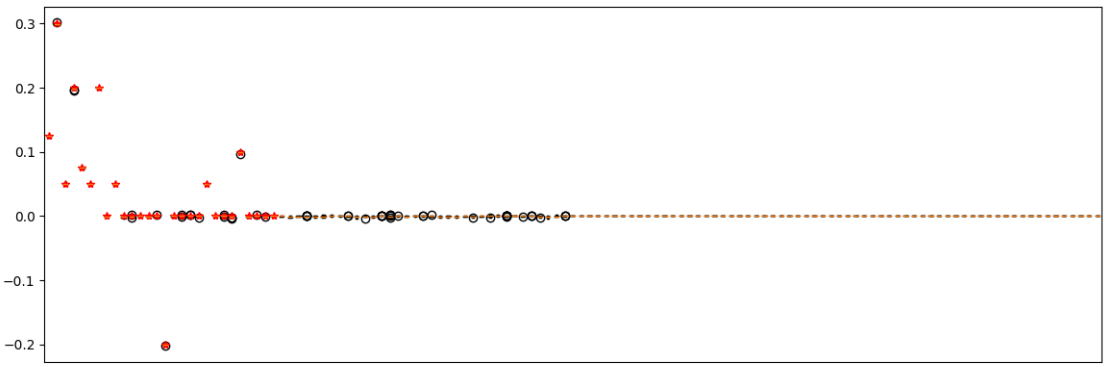


Figure 9.24: Distribution of the parameters of the aggregators of a CI with C3 aggregators, trained on the 62 corners of the unit hypercube with at most 3 coordinates set to 1. The labels are computed by the standard flat model.

It is interesting to notice that a few, well-informed points are sufficient to fully recover the ground-truth model. This is of great interest, as it shows that, in clean contexts, NEUR-HCI models are efficient, as they only require little information to uniquely and consistently identify the real parameters. Moreover, it has the added advantage of learning the model end-to-end rather than locally, avoiding possible bias induced by local elicitation of each module.

This is also encouraging for active learning methods. Indeed, if we noticed some instability when training a given model, and an oracle was available to provide new points, we could request some of these highly informative points to complete the dataset, and thus constrain the model more strongly on area where it was overly too lose due to lack of data points.

It would be interesting to determine the minimal set of points to fully determine an HCI through combinatorial methods, and see whether NEUR-HCI could be trained efficiently on such a small dataset.

9.2.6 Comments and Conclusion

We saw in this section that, when given the right amount of data, and the expert information regarding the hierarchy of the model, NEUR-HCI is able to consistently retrieve the right parameters, both in terms of aggregator weights and of marginal utility. This is coherent with the identifiability result with a fixed hierarchy that we present in Chapter 4.

This is encouraging for working on theoretical results studying the shape of the loss function. Moreover, we see some vulnerability to heavy noise, especially

if datasets are small, and if marginal utilities are required. Some technical work in regularizations and training methods might help in improving these issues, and are to be investigated.

We see also that NEUR-HCI models do not require too large a dataset to train on. In particular, small but informative datasets ensure a good training, and a nicely trained model. We thus believe that NEUR-HCI can be a good alternative to combinatorial methods when trying to build a model hand in hand with an expert, by minimizing the required interactions to:

- asking the hierarchy
- validation of the single model returned after training

9.3 Learning a Model without a Known Hierarchy

In this section, we illustrate how we can train NEUR-HCI models when the hierarchy is not readily available. We thus evaluate whether we still obtain good performance. Moreover, we observe that some of the model's indicators remain intact, making explanation possible even in this setting.

9.3.1 Performance

We experiment on diverse methods for approximating a model with wrong or uninformed hierarchies. Table 9.7 shows different settings, in different dimensions, for testing the models. Our settings include:

- training a flat model to learn data generated by a tree model
- training a tree model to learn data generated by another tree model
- training a linear model to learn data generated by a tree model
- training a DAG-CI model to learn data generated by a tree model
- training a DAG-CI model to learn data generated by a flat general CI
- training a tree model to learn data generated by a flat CI

Unless otherwise stated, both generators and trained models have C2 aggregators. We also work in the regression setting, as we aim to see whether a model can be approached closely by another with different hierarchy.

Note that the tree structure has no influence on the model if the aggregators are linear (that is, any hierarchy can represent any linear model). That is, any

model should be able to learn the linear part of the ground truth model, with the main source of error being that a given hierarchy cannot represent the non-linear parts of a model with another hierarchy.

We notice that the linear model is the one that has the worst generalization error among all models. We also see that trying to learn a tree model using a flat model yields better performance than using another tree, which is understandable, as a flat model can represent more interaction behaviours. On the other hand, we see that the DAG models are the best for approximating a random tree, and that a DAG with C2 aggregator can approximate a general CI well. The DAGs we worked with had a single hidden layer, with the same width as the input layer.

Nonetheless, these results have to be mitigated; indeed, our tree and FM generators were naive, and thus probably biased. Moreover, the tuning of a DAG's hyperparameter has proven tricky, especially when the width of different layers differ greatly. It is thus not excluded that these results may be improved greatly (in the case of DAGs) by some work on different methods for tuning and regularizing, varying the layers' width and the model's depth.

In particular, we saw that, often, many of the CIs of the same layer tended to learn the same weights. This clearly shows the need for working on a regularization method, forcing the CIs to learn different weights, in order to escape the single basin of attraction around a flat model. This also lead to training errors that did not converge to 0., with the model stuck around a local minimum, and a performance that is only slightly better than the flat model.

9.3.2 Stability of the Indicators across the Models

We also notice that, even when training a model with the wrong hierarchy, some indicators (i.e. specific elements of the models that are used for interpretation) remain constant, or stable. The agnosticity of such indicators is highly beneficial for being able to interpret a model even if the given hierarchy is far from the real one.

We notice that, in the large sample limit, we have very good approximations of the marginal utilities of the model, indicating that they seem agnostic from the aggregator's hierarchy, as hinted in the identifiability result. Moreover, we notice that the global Winter values for all attributes seem to remain close for all models trained on close data, regardless of their hierarchy. Figures 9.25 and 9.26, along with those in Annex A.5 display this phenomenon in various situations, where we trained models with random hierarchies on datasets created by a ground-truth model.

We note that, for some cases, there is a bias between the real value of a Shapley or Winter value (depending on the setting) and the learned versions, despite having low variance. This is an interesting phenomenon, which would deserve to be stud-

Hierarchy	5	10	20
Flat / Tree	0.00042 ± 0.00051	0.00051 ± 0.00057	0.00053 ± 0.00060
Tree / Tree	0.00195 ± 0.00130	$0.00088 \pm 0.0.00068$	0.00074 ± 0.00071
LR / Tree	0.00354 ± 0.00254	0.00255 ± 0.00202	0.00202 ± 0.00172
DAG / Tree	0.00021 ± 0.00016	0.00031 ± 0.00017	$0.00046 \pm 2.1 \times 10^{-5}$
DAG / CG	$1.1 \times 10^{-5} \pm 4.3 \times 10^{-6}$	$0.0001 \pm 1.6 \times 10^{-5}$	$7.1 \times 10^{-5} \pm 3.3 \times 10^{-6}$
Tree / Flat	$5.5 \times 10^{-5} \pm 1.3 \times 10^{-5}$	$0.00018 \pm 5.2 \times 10^{-5}$	0.00044 ± 0.00021

Table 9.7: Mean Squared Error in regression setting. Datasets contain 1000 examples generated by 50 random models (random hierarchy and weights). The name of each line is read as *nature of the models trained / nature of the generator models*. The dimension of the models (number of criteria) is given by the column. In the first 4 lines, 50 datasets were generated by random HCIs, each with a random set of weights and hierarchy. We trained flat models (line 1), tree models with random hierarchies (line 2), linear regression, and DAG-CI models over these datasets. The MSE is given over a large testing dataset of $10,000$ examples. The 5th line is learning with DAG-CI data generated by a flat, general, randomly generated CI. The last line is training 50 tree model with different random hierarchies on data generated by randomly generated 2-additive flat models. Each time, 1 model is trained on 1 dataset, the average and standard deviation over all 50 models is given each time.

ied in more details, in particular understanding which context makes a given value be consistently over- or under-valued by other hierarchies. This tends to show that such bias is linked to the hierarchy of the model that generated the data, rather than the ones of the models that were trained on the data. In particular, this bias is not present when the data is generated by a flat model.

When using DAGs, it becomes necessary to generalize the global Winter values, in order to obtain indicators of the importance of a given criterion in the model. We do so by introducing the *generalized global Winter values*, or *DAG-Winter values*:

Definition 15. *Let $k \in M$ a node of the hierarchy. Let $\mathrm{Pa}(k)$ the set of parents of node k. For a node $v \in V$, and $j \in \mathrm{Ch}(v)$, we write $\Phi_j^{\mu_v}$ the Shapley value of j w.r.t. the FM that parameterizes one of its parent v. In other words, $\Phi_j^{\mu_v}$ is the importance of j in the aggregation v, relative to that of the other children of v.*

*Then, the **DAG-Winter value** Ξ_k of node k is computed recursively as:*

$$\begin{aligned} \Xi_r &= 1 \\ \forall k \in M \setminus \{r\}\ \Xi_k &= \sum_{v \in \mathrm{Pa}(k)} \Xi_v \Phi_k^{\mu_v} \end{aligned} \tag{9.2}$$

The philosophy behind it is that, in global Winter values for trees, we multiply the Shapley values along the only path from a node to the root. In a DAG, there

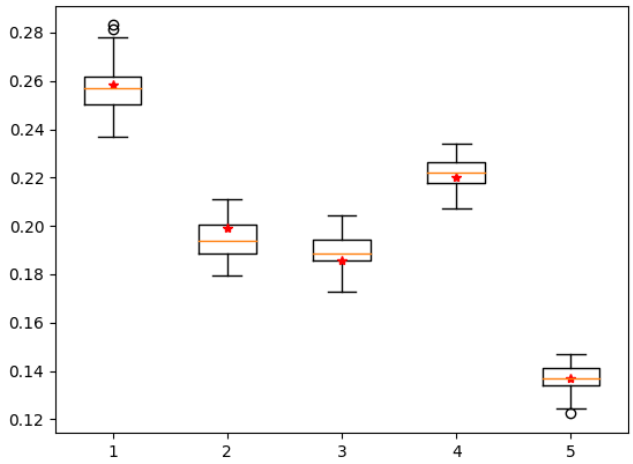


Figure 9.25: Global Winter values for 50 tree models, each with a randomly generated hierarchy, trained on (different) datasets (1000 examples each, dimension 5) generated by the same randomly generated model (random hierarchy and weights).

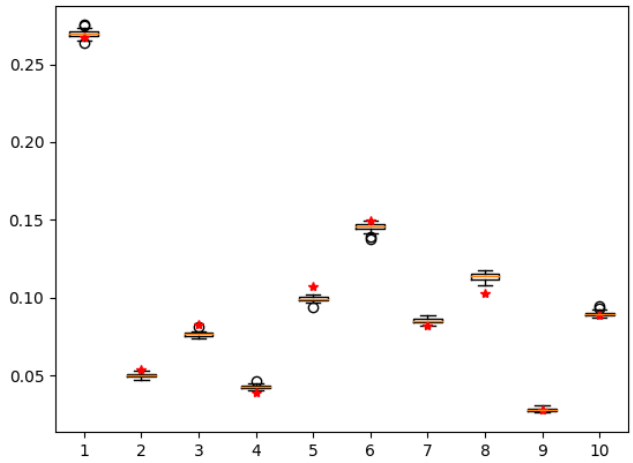


Figure 9.26: Global Winter values for 50 tree models, each with a randomly generated hierarchy, trained on (different) datasets (1000 examples each, dimension 10) generated by the same randomly generated model (random hierarchy and weights).

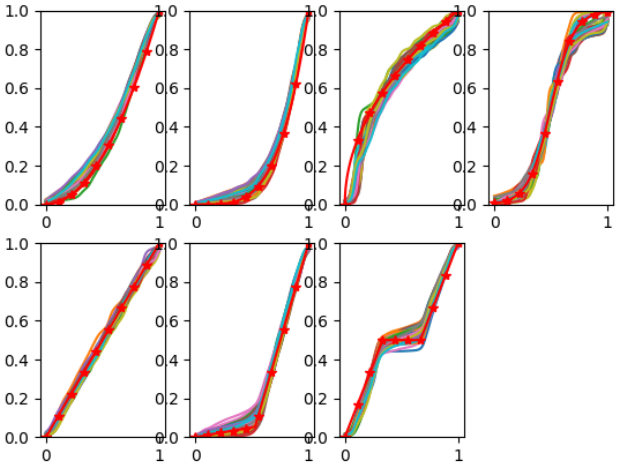


Figure 9.27: Plotting the marginal utilities of 50 models trained in regression setting, 1,000 training examples generated by a flat UCI, and trained on UHCIs with random different hierarchies. The ground-truth marginal utilities are marked with crosses.

might be several paths from a given node to the root; we thus apply the same process on each path and sum them.

Example 37. *Assume the DAG-CI presented in Figure 9.28. We do not detail their respective FM, but we assume that the aggregators have the following Shapley values for their respective children:*

- *Agg.* 5 *has children* $(1, 2, 3, 4)$ *with respective Shapley Values* $(0.5, 0.2, 0.2, 0.1)$
- *Agg.* 6 *has children* $(5, 2, 3, 4)$ *with respective Shapley Values* $(0.1, 0.2, 0.3, 0.4)$
- *Agg.* 7 *has children* $(6, 2, 3, 4)$ *with respective Shapley Values* $(0.2, 0.2, 0.6, 0.)$

Then we can compute:

$$
\begin{aligned}
\Xi_7 &= 1 \textit{(as } r = 7 \textit{ in this case)} \\
\Xi_6 &= \Xi_7 \Phi_6^{\mu_7} = 0.2 \\
\Xi_5 &= \Xi_6 \Phi_5^{\mu_6} = 0.2 \times 0.1 = 0.02 \\
\Xi_1 &= \Xi_5 \Phi_1^{\mu_5} = 0.02 \times 0.5 = 0.01 \\
\Xi_2 &= \Xi_5 \Phi_2^{\mu_5} + \Xi_6 \Phi_2^{\mu_6} + \Xi_7 \Phi_2^{\mu_7} = 0.004 + 0.04 + 0.2 = 0.244 \\
\Xi_3 &= \Xi_5 \Phi_3^{\mu_5} + \Xi_6 \Phi_3^{\mu_6} + \Xi_7 \Phi_3^{\mu_7} = 0.004 + 0.06 + 0.6 = 0.664 \\
\Xi_4 &= \Xi_5 \Phi_4^{\mu_5} + \Xi_6 \Phi_4^{\mu_6} + \Xi_7 \Phi_4^{\mu_7} = 0.002 + 0.08 + 0. = 0.082
\end{aligned}
$$

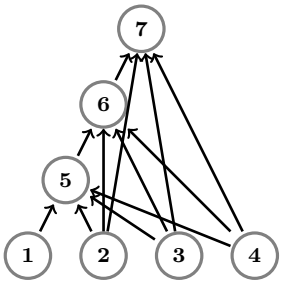


Figure 9.28: DAG used in Example 37

*Note that the sum of the DAG-Winter values of the leaves (*1 *to* 4*) is* 1*.*

We see that this is equivalent to the global Winter value when working with a tree (in this case, there is only a single parent for each node). Nonetheless, this is at the moment an experimental value, as a natural extension of the global Winter values to DAGs, which we use for these tests. It requires some formal work, and axiomatization to see whether it holds the same intelligibility as the Shapley and Winter values in their respective settings.

Despite their experimental nature, we notice that, indeed, these values are stable for many DAGs trained on similar data. Moreover, they also provide good approximations of the Winter values of the generating model, leading us to expect them to have some interesting properties, and serve as a surrogate for interpreting DAG-CI models. Nonetheless, while these values are clearly correlated to the global Winter values of the ground-truth model, we notice a significant bias in some cases (see, for instance, Figure 9.30). These results are illustrated in Figures 9.29 and 9.30, along with Annex A.6.

This bias is interesting as it is consistent across the models (that is, all models undervalue or overvalue a given parameter, despite the dataset being different each time). It would be interesting to study the origins of this phenomenon, given the ground truth hierarchy and the size and width of the DAG network. As for the previous setting, where we learned with purposefully erroneous hierarchies, the biases seem to be correlated to the generator's hierarchy.

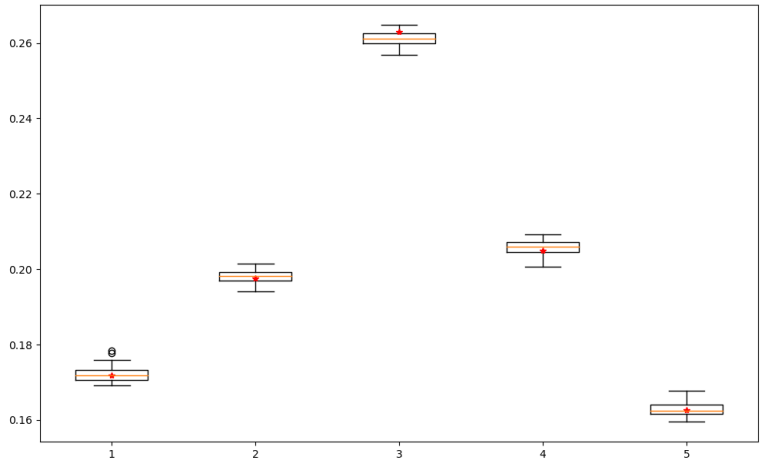


Figure 9.29: DAG-Winter values for 50 DAG models, each with a randomly generated hierarchy, trained on (different) datasets (1000 examples each, dimension 5) generated by the same randomly generated model (random hierarchy and weights).

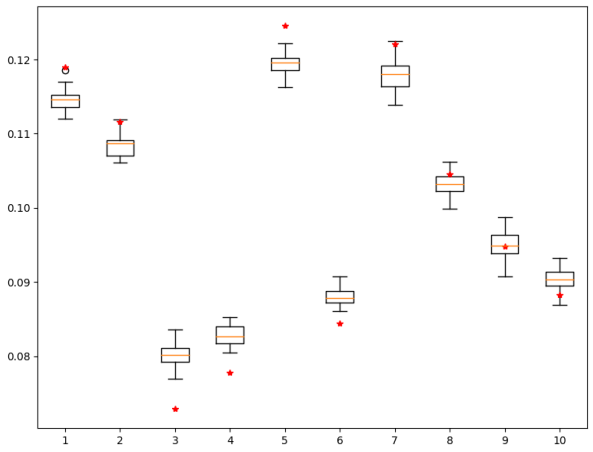


Figure 9.30: DAG-Winter values for 50 DAG models, each with a randomly generated hierarchy, trained on (different) datasets (1000 examples each, dimension 10) generated by the same randomly generated model (random hierarchy and weights). A clear bias is observed between the real and learned values.

9.3.3 Evaluating the Global Winter Values as Weights of a Linear Approximators

It is a known result that the Shapley values are the weights of the linear function that minimizes the quadratic error with a given CI [Grabisch, 2016]. That is, the vector $w = (w_1, ..., w_n)$ that minimizes:

$$\int_{[0,1]^n} (wx - C_\mu(x))^2 \, dx$$

is such that $w_i = \Phi_i^\mu$ for each criterion i.

We study here whether this result still holds empirically for global Winter values (resp. DAG-Winter values). That is, we study whether the global Winter values (resp. DAG-Winter values) are the weights of the linear function that minimizes the quadratic error with a given HCI (resp. DAG-CI). Below, we use the term *importance value* of a criterion with regards to a model to designate either its Shapley value (if the model is a CI), its global Winter value (if the model is an HCI) or its DAG-Winter value (if the model is a DAG-CI).

In order to test this, we randomly generate 50 CI models, 50 HCI models (each with its own random hierarchy), and 50 DAG-CI models. We use each model to generate a dataset of $10,000$ points. We then train, on each dataset, scikit-learn's linear regression model[2], which aims at finding the weights that minimize the residual sum of squares (i.e. quadratic error). We allow the model to fit an intercept. We thus obtain, for a given model Q, a vector $w^Q = (w_1^Q, ..., w_n^Q)$ which we compare to the Shapley values (for CIs), to the global Winter values (for HCIs) and to the DAG-Winter values (for DAG-CIs).

Figures 9.31 to 9.33 show the scatter plot of all the learned weights against the importance value of the same model for the same criterion. That is, if model Q has importance value ϕ_i^Q on criterion i, and if w^Q is the weight vector learned by linear regression on data generated by Q, the dot with coordinates (ϕ_i^Q, w_i^Q) will appear on the plot. A point close to the $x = y$ line means that the learned weight is close to the corresponding importance value.

We clearly see that, in all cases, the weights learned by linear regression fit the importance value of each model very well, as they are all disposed on the $y = x$ line. In particular, the average absolute error between an importance value and the corresponding learned weight is in the range of the machine epsilon.

We can thus expect that the result evoked for CIs at the start of this section still holds for (generalized) global Winter values in the cases of DAG-CIs and HCIs. While these empirical results are no formal proof, they definitely show that these importance values are good candidates for the weights of the optimal linear

[2]https://scikit-learn.org/stable/modules/linear_model.html

approximator of either model. Moreover, they motivate their use as an explanation tool for interpreting NEUR-HCI models.

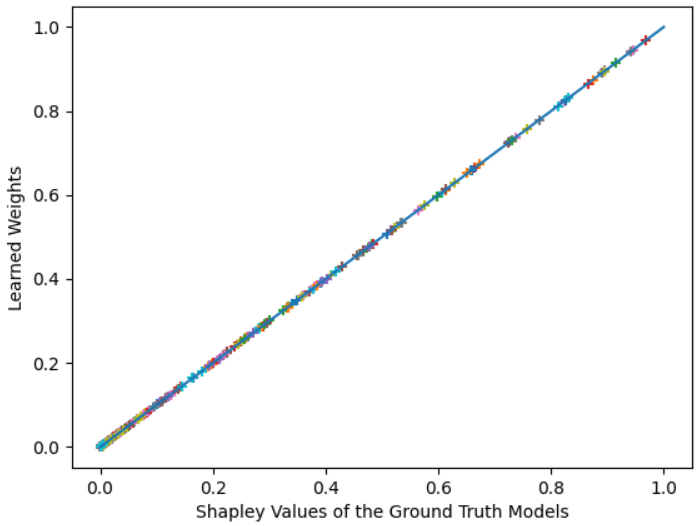


Figure 9.31: Plotting the learned weights of the linear models against the Shapley value of the generator CI model for the same criterion. The plot of $y = x$ is also shown.

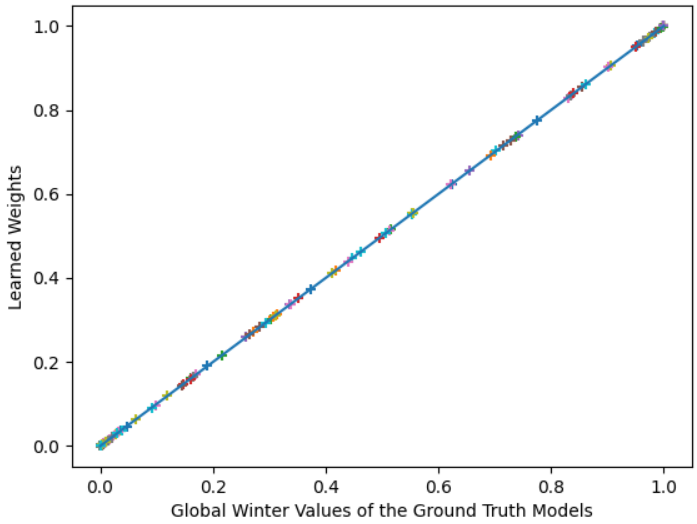


Figure 9.32: Plotting the learned weights of the linear models against the global Winter value of the generator HCI model for the same criterion. The plot of $y = x$ is also shown.

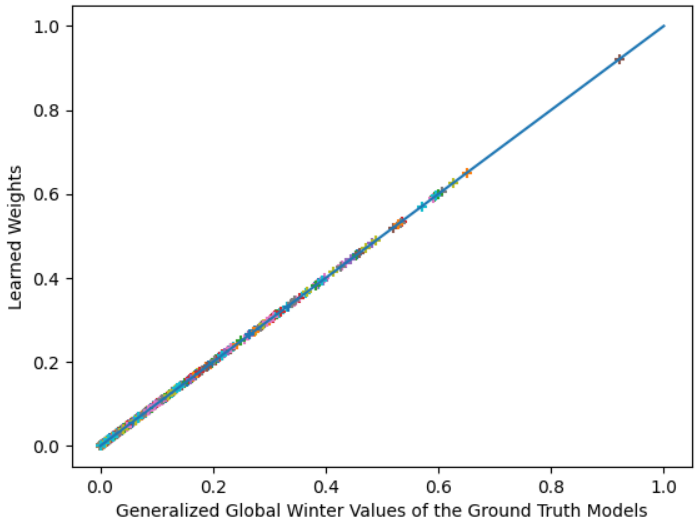


Figure 9.33: Plotting the learned weights of the linear models against the DAG-Winter value of the generator DAG-CI model for the same criterion. The plot of $y = x$ is also shown.

9.4 Real Data

9.4.1 Setting

We assess in this section the robustness of NEUR-HCI on real data, in the classification, regression and ranking settings. The second goal is to investigate the respective impacts of learning marginal utilities and using a hierarchical aggregation. Accordingly, four NEUR-HCI variants have been considered: NCI learns a flat Choquet integral; NCI+U learns a flat CI together with marginal utilities; NHCI learns a hierarchical CI with a tree-structured hierarchy; NHCI+U learns an HCI together with marginal utilities.

NEUR-HCI variants are compared to the following baselines: Multilayer perceptron (MLP) with 1 fully connected hidden layer of n^2 neurons, sigmoid activation function; Logistic regression (in the classification setting); Linear regression (in the regression and ranking settings); Choquistic Utilitaristic Regression[3] (CUR, in the classification setting) [Fallah Tehrani et al., 2014]. The aggregators are 2-additive for improved interpretability and complexity.

[3]As the binarization used with CUR was not available, we ran CUR on the same splits and labels as NEUR-HCI for a fair comparison; this might explain the performance differences w.r.t. the original paper.

We compare on standard monotonic benchmarks, which include *CPU, CEV, LEV, MPG, DenBosch* (DB), *Mammographics* (MG), *Journal*[4], *Boston Housing*[5], *Titanic*[6] and the *Dagstuhl-15512 Arguments Quality* corpus[7] [Wachsmuth et al., 2017]. The last one, reporting the preferences of three decision makers, yields three sub-datasets referred to as *Arguments 1, Arguments 2, Arguments 3* (each one being associated with a single decision maker). A hierarchy is given for the *CEV* and *Arguments* datasets; for all other datasets but *LEV*, the authors managed to build a hierarchy; for *LEV*, no hierarchy was agreed upon (indicated as NA in the results). Overall, the number of features ranges from 4 to 15; the number of examples ranges from 119 to 1728. Ranking datasets are built from classification (respectively regression) datasets, setting that $\mathbf{x}^{(i)} \succ \mathbf{x}^{(j)}$ whenever $y^{(i)} > y^{(j)}$ (resp. $y^{(i)} > y^{(j)} + .05$).

Each feature is associated with its monotonicity, i.e. whether the global score $U(\mathbf{x})$ increases with x_i [8]. All the features and labels in the training sets, were normalized linearly in $[0, 1]$.

The performance indicators are measured as follows. Each dataset is randomly split into an 80% train and 20% test sets; the performance of the model trained from the train set is measured on the test set, and averaged over 1,000 random splits. Each time, all of the methods are evaluated on the same 1000 splits. Depending on the setting, the performance indicator is the misclassification rate, the mean squared error, and the swapping rate in the ranking setting.

9.4.2 Analysis of the Results

Tables 9.8, 9.9 and 9.10 respectively report the NEUR-HCI performances in the classification, regression and ranking settings. The best result for each dataset is displayed in bold [9]. The MLP and NEUR-HCI computational costs are below 5 minutes for each dataset on an Intel i7. The CUR computational cost is not comparable (Matlab implementation).

As could have been expected, MLP generally behaves well; its number of weights, cubic in the number of criteria, however precludes the interpretation of the model. NEUR-HCI behaves on a par with, or better than, the other interpretable models, CUR and linear models.

[4] https://cs.uni-paderborn.de/?id=63916

[5] http://lib.stat.cmu.edu/datasets/boston

[6] https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/ problem12.html

[7] http://argumentation.bplaced.net/arguana/data

[8] The monotonicity is heuristically determined from the dataset, and decreasing criteria are multiplied by -1 to comply with the fact that a Choquet integral is non-decreasing w.r.t. its inputs.

[9] when several results are in bold, this means that the difference is not statistically significant

A first remark is that learning marginal utilities does improve the performance: NCI+U and NHCI+U perform better than their NCI and NHCI counterparts. While this improvement can be naturally explained from the increased depth of the neural architecture and thus the higher complexity of the model, it does not come at the expense of the model interpretation.

A second remark is that hierarchical models outperform flat models on both datasets with an expert hierarchy (*CEV* and *Arguments*). On some other datasets, e.g. *MPG, DG, MB*, the hierarchy adversely affects the performance; as said, this hierarchy is merely guessed by the authors. This suggests that a flat model is preferable to one with an irrelevant hierarchy.

Dataset	MLP	Logistic Reg.	CUR	NCI	NCI+U	NHCI	NHCI+U
CPU	**0.015 ± 0.021**	0.091±0.051	0.024 ± 0.025	0.045±0.039	0.023±0.024	0.030±0.027	0.023±0.026
CEV	**0.004 ± 0.004**	0.110±0.023	0.084±0.067	0.059±0.012	0.051±0.023	0.035±0.009	0.019±0.017
LEV	**0.135 ± 0.021**	0.161± 0.022	0.143±0.0213	**0.136 ± 0.022**	**0.135 ± 0.019**	N/A	N/A
MPG	0.113 ± 0.036	0.090 ± 0.030	0.112 ± 0.099	0.086 ± 0.027	**0.079 ± 0.027**	0.085 ± 0.029	0.082 ± 0.027
DB	0.143 ± 0.069	0.164± 0.071	0.235 ± 0.017	0.139±0.067	**0.132± 0.068**	0.141 ± 0.068	**0.132 ± 0.066**
MG	0.179 ± 0.028	0.196 ± 0.027	**0.166± 0.022**	0.195 ± 0.027	**0.166 ± 0.026**	0.201 ± 0.030	0.181 ± 0.028
Journal	**0.180 ±0.063**	0.250±0.070	0.218±0.086	0.207±0.065	0.197±0.060	0.219±0.065	0.216±0.062
Boston	0.124 ± 0.030	0.145±0.033	0.1360± 0.085	0.127±0.031	0.129±0.032	**0.121±0.032**	0.129±0.031
Titanic	**0.182 ± 0.025**	0.202 ± 0.027	0.185 ± 0.041	0.192±0.0264	0.193 ± 0.027	0.203±0.027	0.194±0.027

Table 9.8: NEUR-HCI, Classification setting: Classification error (average and variance over 1,000 runs).

Dataset	MLP	Linear Reg.	NCI	NCI+U	NHCI	NHCI+U
CPU	**0.0005 ± 0.0016**	0.0022±0.0019	0.0023±0.0032	0.0009±0.0013	0.0026±0.0023	0.0009±0.0011
CEV	**0.0094 ± 0.003**	0.0434±0.0442	0.0437±0.0037	0.0264±0.0027	0.0197±0.0017	0.0176±0.0017
LEV	0.0312 ± 0.0254	**0.0252±0.0029**	**0.0252±0.0031**	**0.0252±0.0029**	N/A	N/A
MPG	**0.0047 ± 0.0008**	0.0089±0.0019	0.0084±0.0018	0.0056±0.0013	0.0091±0.0018	0.0057±0.0012
Journal	0.0410 ± 0.010	0.0524±0.0128	0.0631±0.0127	**0.0385±0.0112**	0.0629 ± 0.0127	0.0391 ± 0.0117
Boston	0.0079 ± 0.0030	0.0174±0.0038	0.0157 ±0.0037	**0.0072±0.0023**	0.0151 ± 0.0033	0.0077 ± 0.0023

Table 9.9: NEUR-HCI, Regression setting: Mean square error (average and variance over 1,000 runs)

Dataset	MLP	Linear Reg.	NCI	NCI+U	NHCI	NHCI+U
CPU	**0.0005 ± 0.002**	**0.0006 ± 0.003**	0.0007 ± 0.003	**0.0006 ± 0.003**	0.0009 ± 0.003	0.0010 ± 0.004
CEV	0.0174 ± 0.012	0.0642±0.011	0.0243±0.005	0.0099±0.002	0.0165±0.004	**0.0088±0.003**
LEV	**0.0178 ± 0.025**	**0.0179±0.023**	**0.0178 ±0.024**	**0.0177±0.023**	N/A	N/A
MPG	**0.0613 ± 0.012**	0.0642±0.011	**0.0610±0.011**	**0.0612±0.011**	0.0633±0.012	0.0621±0.011
DB	0.1355 ± 0.0796	0.1257±0.079	0.1216±0.081	**0.0942±0.069**	0.1231 ± 0.092	**0.0962 ± 0.081**
MG	0.2601 ± 0.046	0.2661±0.047	0.2668±0.045	**0.2381±0.037**	0.2701±0.052	0.2446 ±0.036
Journal	0.1801 ± 0.064	0.1802±0.065	0.1761±0.063	0.1838±0.066	**0.1711±0.063**	0.1889±0.065
Boston	**0.0659 ± 0.016**	0.0790±0.014	0.0790±0.015	**0.0669±0.012**	0.0752 ± 0.014	0.0681 ± 0.014
Titanic	**0.1521 ± 0.027**	0.1651 ± 0.029	0.1632 ±0.028	**0.1533 ±0.028**	0.166 ± 0.028	**0.1542 ± 0.029**
Arguments 1	0.0157 ± 0.015	0.0195±0.016	0.0145±0.012	**0.0141±0.012**	**0.0141±0.012**	**0.0140±0.012**
Arguments 2	0.0588 ± 0.028	0.0653±0.031	0.0644±0.028	0.0581±0.027	**0.0572±0.027**	**0.0572±0.028**
Arguments 3	**0.0740 ± 0.039**	0.0941±0.042	0.0783±0.040	0.0784±0.040	**0.0761±0.039**	**0.0771±0.041**

Table 9.10: NEUR-HCI, Ranking setting: percentage of mis-ordered pairs (average and variance over 1,000 runs)

0.008±0.001	0.008±0.002	0.013±0.003
0.0±0.0	0.016±0.006	0.017±0.007
-0.007±0.006	0.001±0.001	0.131±0.005
0.03±0.011	0.168±0.008	0.004±0.004
0.121±0.006	0.025±0.008	0.129±0.003
0.047±0.005	-0.013±0.007	0.025±0.007
0.08±0.006	0.169±0.009	0.03±0.007

Table 9.11: CPU dataset (regression setting): Mean and standard deviation of the model parameters (Möbius values) over 100 runs.

9.4.3 Stability

In terms of model identification, the solution trained from a given dataset along independent runs is very stable, i.e. the parameters always converge towards the same values, as is observed on artificial datasets. Table 9.11 reports the variance over 100 runs of the 21 model parameters (the Möbius representation [Grabisch and Labreuche, 2008]) on the CPU dataset.

The model stability makes room for the easy visual inspection of the model. Indeed, a 2-additive model can be represented in the 2D plane, with the importance of the i-th criterion (respectively the synergy between i-th and j-th criteria) depicted as the color of the (i, i) (respectively (i, j) and (j, i)) squares: the darker the more important. As illustrated on Fig. 9.34, the most important criteria in the CEV domain are the safety rating (criterion 6) and to a lesser extent the passenger capacity (criterion 4); criteria 4 and 6 also happen to have a strong synergy. The buying price and the security rating also have a significant synergy. It is emphasized that this straightforward visualization enables the decision maker to instantly check the trained model; the estimated synergies can be confronted to the expectations and if needed the model can be revised by augmenting the dataset, adding samples to illustrate the desired effects.

The performance of Neur-HCI in terms of classic error metrics (classification loss, mean squared error) has been evaluated on diverse multicriteria datasets, against several other models, including a similarly sized ANN. The results, which can be found in [Bresson et al., 2020c], show that Neur-HCI performs globally well, and can beat the unconstrained ANN on several datasets. We will here evaluate its robustness and stability; that is, check whether the parameters of several learned models are similar when trained on the same dataset, or slightly altered versions of the same dataset. Such alterations involve adding noise, or training on different parts of the dataset.

In the artificial case, we used an HCI with marginal utilities to generate examples, then trained our models on such data. In practice, every trained model

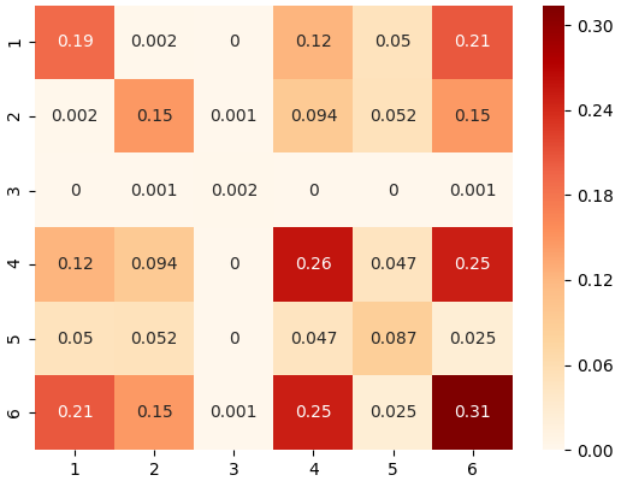


Figure 9.34: CEV dataset: Visual inspection of the trained model, showing the importance of criteria 4 and 6, of their synergy, and to a lesser extent the synergy of criteria 1 and 6.

converged to the real one in the large sample limit (fewer than 100 samples were enough in a 6-dimensional case).

We illustrate here the stability of Neur-HCI on the auto-MPG dataset[10]. It holds 292 alternatives described by 7 criteria. Fig. 9.35 shows the parameters values learned by 100 Neur-HCI networks in the regression setting, with a single CI, each being trained on a different split of the dataset (80% of the dataset was randomly chosen and used to train each model). We use a more compact version of the parameters called their Möbius representation. One can see that the parameters values remain similar for all models, indicating a strong influence of criterion 4, followed by criteria 3 and 5. The marginal utilities are also very similar for all models, as shown in Fig 9.36. This offers a first empirical confirmation of the stability of Neur-HCI, as we obtain close models for close training sets.

Figures 9.37 and 9.38 show the stability of the interaction indices of the parameters on the CEV and LEV datasets respectively. We also notice an important stability.

[10]collected from http://archive.ics.uci.edu/ml/index.php

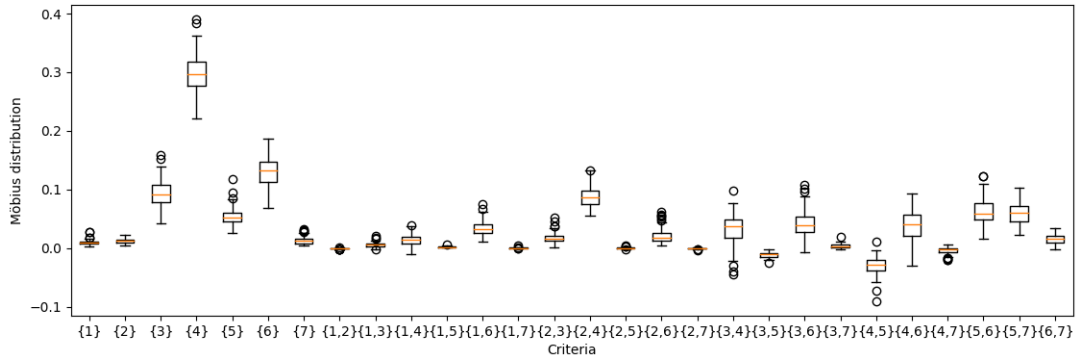


Figure 9.35: Learned parameters (Möbius values) of 100 flat models trained on different splits of the MPG dataset.

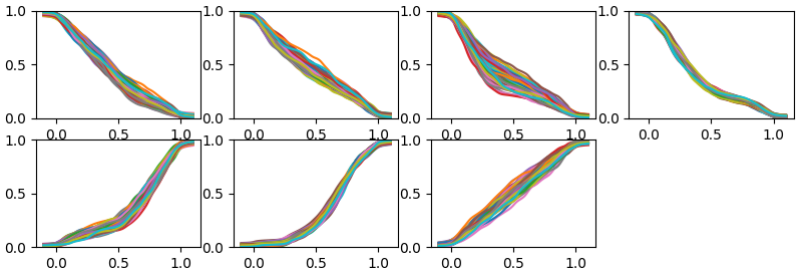


Figure 9.36: Learned marginal utilities of the models from Figure 9.35, trained on the MPG dataset.

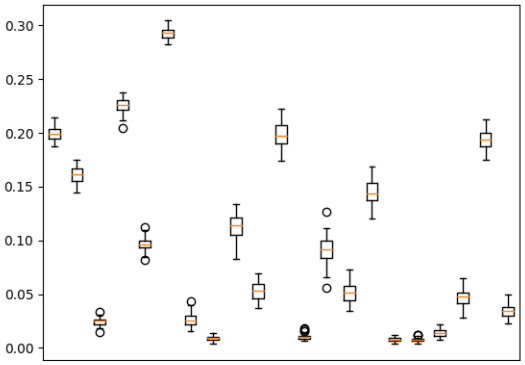


Figure 9.37: Learned parameters (interaction indices) of 100 flat models trained on different splits of the CEV dataset.

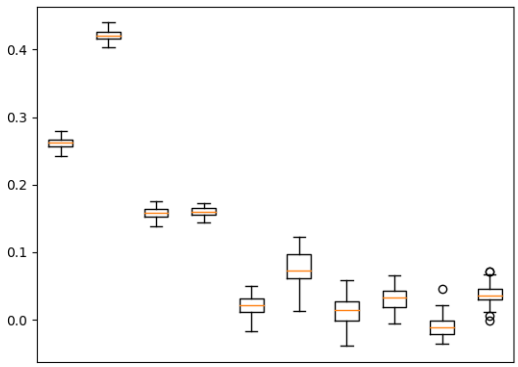


Figure 9.38: Learned parameters (interaction indices) of 100 flat models trained on different splits of the LEV dataset.

9.5 Training Time

We present below the time it took to train a model, depending on the type of its aggregators, dimension, and hierarchy. We tested on:

- Flat models, which maximizes the number of parameters, but minimizes the number of aggregation steps
- Randomly generated trees, which gives an average between both values
- Binary trees, which maximizes the number of aggregators, but minimizes the number of parameters. We look at two types of binary trees:
  - chain binary tree, i.e. a binary tree where each non-leaf node has two children, one of which is a leaf (see Figure 9.39)
  - quasi-balanced binary tree, i.e. a binary tree where, for each non-leaf node k, the number of descendants to the right of k differs of at least one with the number of descendants to the left of k, (see Figure 9.40)

The results for all cases are reported for aggregators of types C2, C3 and CG, respectively in Tables 9.15, 9.16 and 9.17, on an Intel i5 CPU. Table 9.12 through 9.14 show the number of parameter for each model (averaged over 50 models in the case of the random trees). For non-random models, we had very little variance, and thus only display the mean time.

We notice that the number of parameters of the model is not the only criterion at play in estimating the time it takes to train a model. Indeed, while the flat

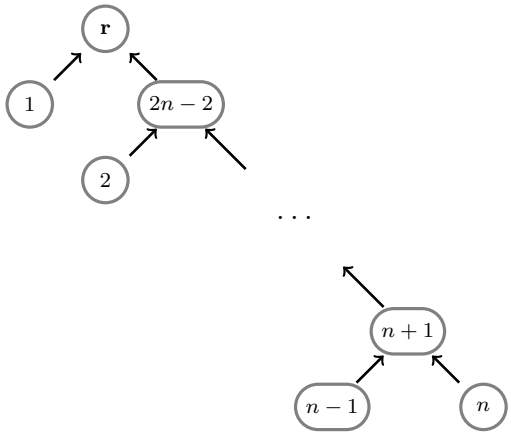


Figure 9.39: Chain-binary tree with n leaves

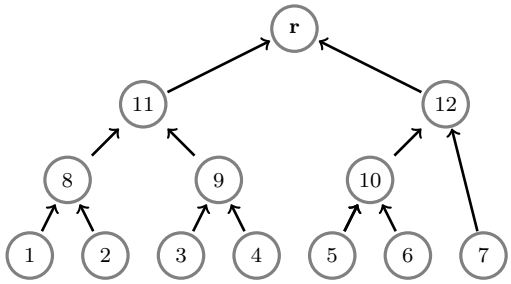


Figure 9.40: Quasi-balanced binary tree with 7 leaves

models are clearly the longest to train in all settings, a random/quasi-balanced hierarchy is still quicker to train than a binary one. This seems to indicate that there is a tradeoff between the number of aggregators and that of parameters. Nonetheless, both are very quick to train when compared to a flat model (which becomes untractable for too large a dimension). We notice no significant difference between the chain-binary model and the quasi-balanced one; nonetheless, it seems that the latter could benefit from parallelization of its computations, given that many nodes are not dependant upon each other.

Moreover, we see that, for tree networks, the time seems linear w.r.t. the dimension. This is clearly understandable for binary models. Indeed, all of the binary aggregators have 4 parameters, meaning that the binary trees have $4(n-1)$ parameters. The fact that we notice the same linearity for random trees seems to point at the fact that our tree generator was biased towards smaller aggregators, making the number of parameters more dependant on the number of aggregators than on their size.

	3	5	10	15	25	50	100
Flat	9	25	100	225	625	2500	10000
Binary	8	16	36	56	96	196	396
Random	8.34	17.67	41.28	64.62	112.21	233.33	477.29

Table 9.12: Average Number of Parameters of the Models, with C2 aggregators.

	3	5	10	15	25	50	100
Flat	17	105	1060	3865	$19,025$	$159,300$	$1,303,600$
Binary	8	16	36	56	96	196	396
Random	10.79	28.42	79.17	135.7	267.44	547.03	1214.81

Table 9.13: Average Number of Parameters of the Models, with C3 aggregators.

	3	5	10	15	25	50	100
Flat	8	32	1024	$32,768$	3.3×10^7	1.1×10^{15}	1.3×10^{27}
Binary	8	16	36	56	96	196	396
Random	8.0	17.24	39.72	66.08	120.6	256.88	559.68

Table 9.14: Average Number of Parameters of the Models, with CG aggregators.

	3	5	10	15	25	50	100
Flat	38	52	104	187	426	1502	5682
Binary C.	58	104	222	336	553	1143	2193
Binary B.	57	103	214	327	549	1104	2211
Random	52 ± 9	81 ± 12	172 ± 17	233 ± 14	393 ± 18	781 ± 34	1587 ± 57

Table 9.15: Training time (seconds) of diverse HCIs with 2-additive aggregators. Lines are the hierarchies, columns are the dimensions. Trained for 1000 epochs on 500 data points. These values are represented in Figure A.55.

	3	5	10	15	25	50	100
Flat	98	538	6808	31304	240159	ND	ND
Binary C.	81	162	327	496	831	1649	3328
Binary B.	80	148	324	493	827	1672	3338
Random	86 ± 8	201 ± 53	673 ± 451	878 ± 445	1435 ± 409	3177 ± 643	6842 ± 591

Table 9.16: Training time (seconds) of diverse HCIs with 3-additive aggregators. Lines are the hierarchies, columns are the dimensions. Trained for 1000 epochs on 500 data points. ND values mean that the training time was too large to realistically train. These values are represented in Figure A.57 and A.56.

	3	5	10	15	25	50	100
Flat	32	458	101	557	514690	ND	ND
Binary C.	43	75	160	234	392	771	1520
Binary B.	42	73	161	238	400	818	1638
Random	38 ± 5	69 ± 6	134 ± 10	196 ± 14	319 ± 14	644 ± 29	1276 ± 25

Table 9.17: Training time (seconds) of diverse HCIs with general aggregators. Lines are the hierarchies, columns are the dimensions. Trained for 1000 epochs on 500 data points. ND values mean that the training time was too large to realistically train. These values are represented in Figure A.58 and A.59.

Part VI

Perspectives and Conclusion

CHAPTER 10

PERSPECTIVES AND CONCLUSION

10.1 Conclusion

Our work focuses on a class of MCDA decision models called UHCI; these models allow to represent an interpretable, highly constrained decision process. This permits to build decision models which are easily validated by a field expert, and can, in turn, be trusted for applications in safety-critical contexts. Nonetheless, traditional methods for building such models often suffer several bottlenecks and drawbacks, such as:

- the heavy need of preferential information obtained from a human DM
- the need to build the model element by element, leading to local constraints that lead to global inconsistencies

Our first contribution [Bresson et al., 2021], presented in Part III, shows that, under mild assumptions, a UHCI model has uniquely defined interpretable parameters. In other words, if two UHCIs give the same score to any two alternatives in the alternative space, then we know them to have:

- the same marginal utilities: which is crucial to determine which values over a given attribute give the highest or lowest criterion-wise satisfaction
- the same hierarchy: which indicates the underlying structure of the aggregation
- the same weights for each aggregator: which determine the relative importance of each criterion, along with the way they interact with each other

This identifiability thus allows a single interpretation to be drawn from observing a model, which in turns encourages trust in the said interpretation. This trust is necessary for a decision maker to accept the model's aid in safety-critical decision contexts.

The bulk of this proof is based on studying the UHCI composed of a piecewise-linear HCI and of piecewise-C^1 marginal utilities is a piecewise-C^1 function. Our proof is based on studying the frontiers between each C^1 regions. We develop two algorithms:

- one for recovering the separation frontiers of a given UHCI from its hierarchy, aggregators and marginal utilities;
- one for recovering the hierarchy from the separation frontiers.

Through highlighting this relation between the structure of the UHCI as a piecewise-C^1 function and the model's parameters (hierarchy included), we show that the hierarchy is unique. We can then go on to prove that the rest of the parameters can be recovered through evaluation in specific points, allowing us to complete the proof.

This result thus encourages our second contribution [Bresson et al., 2020c, Bresson et al., 2020b], introduced in Part IV: a neural-networks framework called NEUR-HCI. NEUR-HCI permits to recover, from data, the parameters of an UHCI (including its marginal utilities), given as little expert knowledge as possible (namely, we require an expert-provided hierarchy).

This framework is an important result to the field, as previous methods for learning similar models were restricted to flat CIs (possibly fitted with marginal utilities). NEUR-HCI uses a modular approach, by representing each of the key components of the UHCI by a small multilayer perceptron, called a *module*, then assembling the modules along the pre-defined hierarchy. Each type of module is designed to represent a specific type of function. The existing modules as to now include:

- Marginal utilities;
  - monotonic.
  - bitonic.
  - a selector module for automatically chosing among the other types.
- Choquet integral-based aggregators;
  - 2-additive CIs;
  - a subset of the 3-additive CIs;

– general CIs.

Each of this module implements by design all of the constraints expected from the function it represents. Most notably, these include monotonicity and normalization. Moreover, they can represent any of the functions of their respective classes, making them relevant for learning such classes.

Once the UHCI is built from the said takes advantage of the backpropagation algorithm to optimize all modules at the same time, effectively learning the model from acquired data. It is important to note that the thus obtained UHCI is formally valid. As a the neural structure is only a tool for learning; once trained, the model can be rewritten as a simple mathematical formula, which is much lighter to use and store, allowing for uses in resource-scarce environments, such as embedded systems. Moreover, the native intelligibility of the UHCI allow for an end user to quickly validate (or invalidate it) prior to using in run-time. The formal guarantees on the model ensure that it satisfies all of the behavioural constraints expected from an MCDA model, making it suitable for decision-aiding in a safety-critical context.

We have shown several interesting empirical results in Part V:

Performance: NEUR-HCI performs well in several learning settings. It achieves good performance on both real and artificial dataset, in regression, binary classification, and pairwise-preference learning. We see that an informed hierarchy allows better performance than a random one, with the flat model as an in-between, offering fairly good performance when no hierarchy is available.

Stability: the thus-learned models are stable, exhibiting robustness to noise and random variability in a dataset. This encourages trust, as models trained on close datasets will have close interpretations. Moreover, we see that indicators such as the hierarchical Shapley values and marginal utilities remain consistent across models with different hierarchies, with the most notable defect being some bias on the Shapley values. This is encouraging, as it shows that, even if the expert-given information or data is incomplete, or incorrect, the model can still be interpreted along important behavioural aspects. These results were also presented in [Bresson et al., 2020a].

Generalization of the Global Winter Value: We propose the DAG-Winter value as a way to generalize the global Winter value to non-tree DAG structures. We show that these indicators present the interesting property of being the optimal weights (w.r.t. the mean squared error) for approximating a DAG-CI with a linear model. We also see that these indicators are stable across models when training, making them relevant values for estimating the relative importance of each crite-

rion.

This work thus effectively strengthens the bridge between MCDA and ML. More generally, it encourages machine learning approaches on intelligible, constrained models, when the applications requires trust and formal guarantees.

We also provide an analysis of the training time depending on the hierarchy of the model, showing that a hierarchy can significantly reduce training time in high dimensions.

10.2 Perspectives

This works opens the way to a wide array of potential future research, both theoretical and technical.

10.2.1 Background and Formal Work

A large amount of work could be dedicated to the study of the representation power of the (U)HCI and DAG-CI models. A classic metric for expressivity is the VC dimension, which could be evaluated on UHCIs, in the wake of the work presented in [Hüllermeier and Fallah Tehrani, 2012] that aimed at bounding it for a simple CI.

Moreover, it would be interesting to study to what extent the additivity and hierarchy-induced constraints affect that representation power. For instance, the following questions might benefit from such analysis:

- how well can a 2-additive CI approximate a higher-additivity/general CI ?
- how well can a flat model approximate any HCI of the same dimension ?
- how well can a $0-1$-based 3-additive CI approximate any 3-additive CI ?
- what orders of sizes are needed for a DAG-CI to be a good approximator of any HCI ?

Answering these questions allow to evaluate how well a model of a certain class can learn a function of a more complex class. This, in turn, is useful for characterizing the set of decision functions that can and cannot be represented under certain complexity assumptions, allowing to automatically switch to a higher-complexity class when needed.

Another strong result would be an extension of the theoretical identifiability result presented in Part III. Indeed, this result only ensures the unicity of the

parameterization of a model defined over the entire, continuous, hyperspace $\mathbb{R}^n$. This result could be greatly reinforced in many ways.

First of all, one could wonder whether this result holds for a model defined on a specific, discrete, or even finite, dataset, which is more realistic from a machine learning point of view. If so, we could evaluate a minimal set of examples, then use active learning methods for targeting specific examples.

Secondly, it would be relevant to evaluate whether two models with very different parameterizations can have very similar outputs for all points in the input set. This is also crucial for machine learning, as it would ensure that, given two very close datasets, we would learn models with very close parameterizations, thus giving a theoretical guarantee over the robustness of the model. This would come as a complement of the empirical stability that we have observed in Part V, where noise only affects the learned model to a controlled extent.

The effect of the hierarchy is also an interesting topic to study; can two models with different hierarchies be ϵ-close everywhere ? If so, do the hierarchies have to be "close" w.r.t. a given metric ? In particular, if two hierarchies have a given sub-tree in common, can we hope to have a similar behaviour among the shared nodes ? These questions could allow to characterize even better the behaviour of the HCI models, reinforcing the formal guarantees available and, in turn, user trust in these models.

Another important point pertains to theoretical learnability. We have seen that there is unicity of the model, but this is merely a first step in establishing accurately the behaviour of the loss function. If we can prove the unicity of the basin of attraction in the loss function, which is hinted by the experimental results, then we can further reinforce the trust and learning processes, and guarantee additionnal robustness on the learned models. We could also try and elicit minimal sets of point to train a model, in the perspective of PAC learnability [Valiant, 1984].

We could also work on the global Winter values for HCIs. We have already observed that they were quite stable, even across hierarchies. We also noticed that the generalized version for DAG-CI also shows the same consistency. It might thus be interesting to study formal properties of these indicators, and to what extent some results that are true for Shapley values and a flat CI generalize to the tree and DAG versions. These include:

- is the global Winter for Tree (resp. DAGs) the best linear approximator of an HCI (resp. DAG-CI) ? We only showed empirically that it seems to be the case, a formal proof would nicely complete this observation;

- can the (generalized) global Winter values be axiomatized in the same way as the Shapley values already are ?

- can the interaction index of a CI be generalized, in the same way, under reasonable axiomatics, to HCIs and DAG-CIs ?

Moreover, we have seen that the values of such global Winter values were sometimes biased when training on a different hierarchy. While this can be influenced by our random generators for hierarchies and aggregator, which is naive and probably itself biased, the study of the effect of a given hierarchy on such values could help building hierarchy-agnostic indicator for interpreting the model. In the same direction, it would be interesting to evaluate the UHCI model in the framework of intelligibility [Audemard et al., 2021], i.e. what informations can be derived from the model

All in all, this work would help understand the Choquet integral and its hierarchical extensions better, allowing to yield more guarantees for decision models based on such aggregators. Such models could also be applied in contexts which benefit from this graph structure, such as causal learning, in which the causality relations between several variables are learned from data.

10.2.2 Experimental and Implementation Work

An obvious extension of Neur-HCI would be to be able to learn the hierarchy from data. Indeed, at the moment, the tree-structure is supposed to be given by an expert. We have tried a method based on building a DAG, then pruning edges in order to recover, in the end, a treelike structure (by ensuring all node have a single remaining outgoing edge). Nonetheless, this method failed, as the model seemed always to converge towards a flat model. This corroborates the fact that the flat model seems to be a good approximator of a tree-like model. Methods based on evolutionary programming, such as those from the field of neuroevolution, focus specifically on tuning the architecture of a neural network. In particular, an approach [Chen and Huang, 2019] based on genetic algorithms and linear programming aims at learning such hierarchies. It would be interesting to try it in concert with Neur-HCI and check the performance of this approach.

Following our proof of identifiability, we could also try and learn an HCI's separation frontiers between its affine regions, and thus rebuild the hierarchy from there. It would be interesting to study the amount and quality of data required. In particular, we have seen that a hand-picked sample of points allows to identify the parameters of an HCI ; an active learning setting thus seems a good idea to collect data for this method.

We could also work on developping neural modules for specific constrained aggregators. One could think, for instance, about models such as OWA, WOWA [Beliakov, 2018] or CIs with higher levels of additivity. We could also generalize to

multilinear models, in which the min terms of the CI are replaced by a product. It would be relevant to then study hierarchical multilinear models.

On the other hand, it would also be reasonable to try and build multi- dimensionnal "marginal" utilities, to improve the representation power of strictly-decomposable models, while maintaining the constrainedness and interpretability. For instance, we could require a satisfaction function $u_{i,j}$ over criteria i and j, such that $u_{i,j}$ is single-peaked w.r.t. x_i, but where the optimal value x_i^* is a non-decreasing function of x_j. This would help deal with cases such as that from Example 2 in Section 2.2.2.2, where the satisfaction over the heart rate is a single-peaked function, with the value at the peak being a non-increasing function of the age. This rejoins some work on the GAMI-Nets [Yang et al., 2021].

Another extension regards the types of models that can be elaborated from the presented modules. We have presented Trees and GANs, but we could imagine an intermediate model. The latter would be a set of HCI $Q_1, ..., Q_m$, each with a different hierarchy, and possibly with different sets of criteria $S_1, ..., S_m$. The output of each HCI is then aggregated by a weighted sum, such that the global model $\tilde{Q}$ can be written as:

$$\tilde{Q}(\mathbf{x}) = \sum_{i=1}^{m} w_i Q_i((x_j,\ j \in S_i)) \tag{10.1}$$

with, $\forall i \in [1, ..., m]\,,\ w_i \geq 0$ and $\sum_{i=1}^{m} w_i = 1$. We thus establish a sub-class of the GAI model, which is a convex sum of HCI. This can be seen as an ensemble method, which would generalize the HCI, while remaining strongly constrained, and interpretable so long as the number of HCIs remains bounded.

There is also work in MCDA on the distinction between statistical correlation (how two variables tend to evolve w.r.t. one another) and preferential interactions (e.g. redundancy, synergy). Methods for decorrelating variables have been studied (e.g. principal component analysis), trying to obtain new, uncorrelated latent features. Neural approaches also exist, such as AgnoS [Doquet and Sebag, 2020], which is based on a type of neural networks called an autoencoder, to build such latent variables. It would be interesting to try and combine both approach by connecting a NEUR-HCI network on the output of such a "decorrelator". Then, the parameters of both models would be learned together; on the one hand, the NEUR-HCI model would learn to solve the task at hand (regression, classification...) while the decorrelator network would make the different criteria as independant as possible. Studying the latent criteria might yield some interesting property of the underlying phenomena.

In terms of learning methods, it seems that NEUR-HCI models could greatly benefit from parallelization of calculations at training time; indeed, when working

with trees, parallel branches can be computed independantly until they meet at their common ancestor. We could also work on adapting NEUR-HCI models to online learning, allowing to extract information from a continuous data stream rather than a fixed dataset. Improvements to the learning pipeline also includes better tuning of the hyperparameters, especially on DAG-CIs, by adapting the learning rate to the size of the aggregator, along as their position in the network. Regularizations could also help the model converge nicely, forcing it to diversify the behaviour of its different neurons (as this is an issue that was noticed when training DAG-CIs).

Finally, the ability of NEUR-HCI to learn significant and meaningful phenomena about the data allows us to hope that a given model can be adapted for a task different than the one it was trained to (for instance, train a model on classification data, then use it for pair-wise preference elicitation). The field called transfer learning specializes in this kind of method, and could help render NEUR-HCI models versatile for diverse MCDA settings.

10.3 Closure

In this thesis, we contributed to the bridging of two fields, by using ML methods to learn MCDA models. We showed that our attempts were successful, making ML ever closer to being used in settings where trusting the models is of the essence.

We have also proposed several axes for future work. Researching along those lines seems highly relevant to us, helping both fields benefit from each other, as the strengths of one can compensate some the other's flaws.

Acknowledgements

This thesis could not have been achieved without the help of a number of people. Now is the time to thank them again for their help and contributions.

First of all, I cannot thank enough my supervisors Johanne and Christophe, and advisors Michèle and Eyke, for providing invaluable help during the entirety of these three years. Their advices and guidance, both in terms of scientific expertise and in more transversal fields, made this work possible.

I am also grateful to the reporters Patrice Perny and Andrea Passerini for accepting to review my thesis, and to the examiners Krzysztof Dembczyński and Hendrik Blockeel for participating in the this Ph.D.'s jury.

I also thank my fellow Ph.D. students, both in Thales and in LISN, in particular Adam, Armand, Douae, Erwann, Paul and Quentin. Casual conversations around a coffee were always a good time, and more serious, scientific talk allowed me to broaden my knowledge to fields I would probably not have explored otherwise. Facing this Ph.D. side by side with such comrades made the difficult moments much easier to overcome.

I finally thank my colleagues in both supervising entities, for their advice and encouragements, as well as their friendliness and eagerness for sharing their expertise.

APPENDIX A

FIGURES FOR THE EMPIRICAL RESULTS

A.1 Figures: fixed hierarchy large sample limit

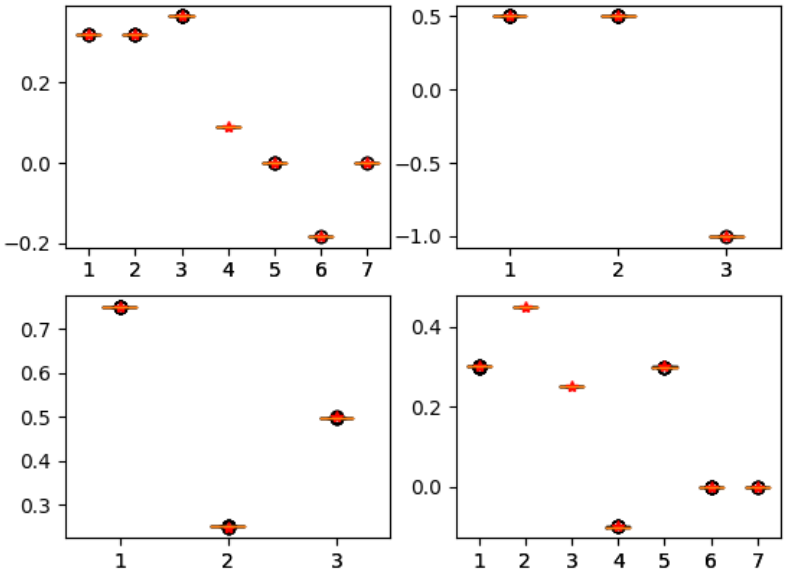


Figure A.1: Distribution of the parameters of the aggregators of a CI with CG aggregators, 10,000 training examples generated by an HCI with 2-additive aggregators. 50 models trained in regression setting. Notice that the parameters of the sets with more than 2 elements are close to 0, as expected since the data is 2-additive.

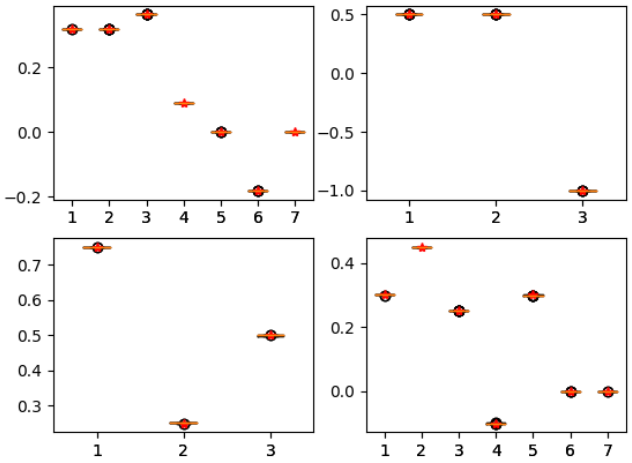


Figure A.2: Distribution of the parameters of the aggregators of a CI with C3 aggregators, 10,000 training examples generated by an HCI with 2-additive aggregators. 50 models trained in regression setting.

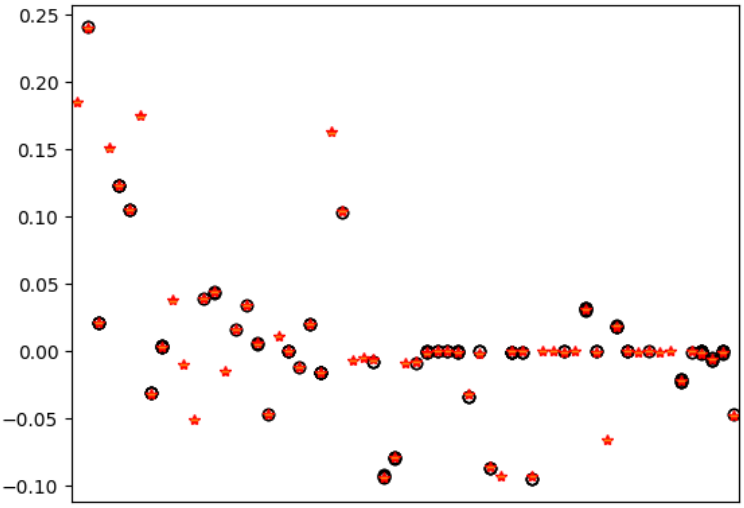


Figure A.3: Distribution of the parameters of the aggregators of a CI with C3 aggregators, 10,000 training examples generated by an HCI with C3. 50 models trained in regression setting.

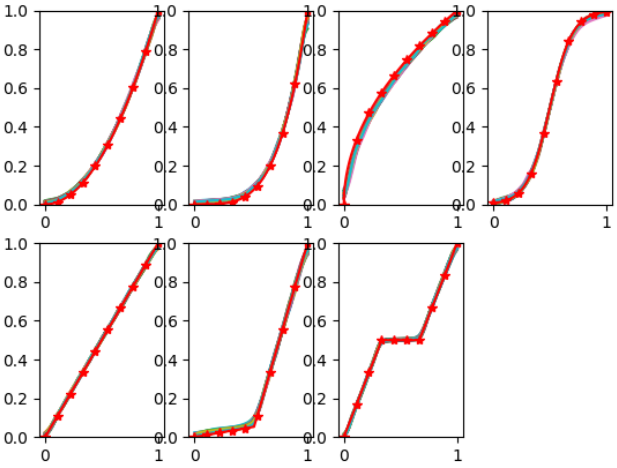


Figure A.5: Marginal Utilities of 50 flat models trained on 1000 artificial datapoints generated by the standard flat model. The ground-truth marginal utilities are represented by the red stars.

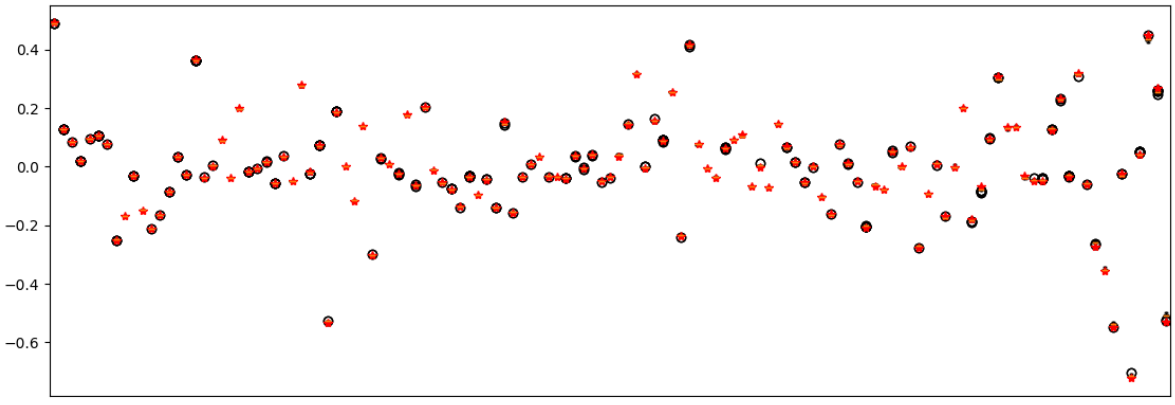


Figure A.4: Distribution of the parameters of the aggregators of a CI with CG aggregators, $10,000$ training examples generated by an HCI with general-additive aggregators. 50 models trained in regression setting.

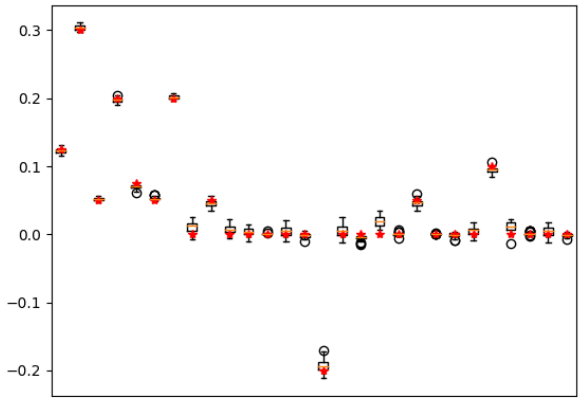


Figure A.6: Distribution of the parameters of the aggregator of a CI with C2 aggregators, 10,000 training examples generated by a CI with general-additive aggregators. 50 models trained in binary classification setting.

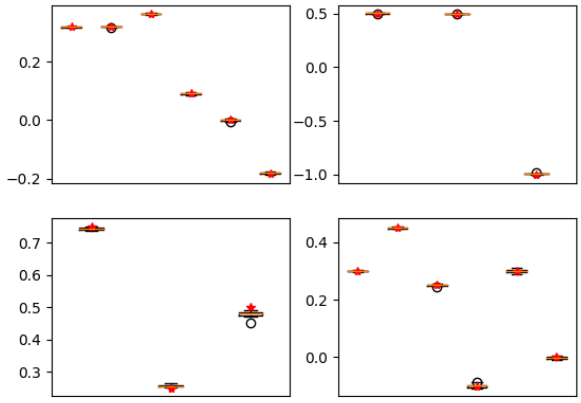


Figure A.7: Distribution of the parameters of the aggregators of an HCI with C2 aggregators, 10,000 training examples generated by an HCI with general-additive aggregators. 50 models trained in binary classification setting.

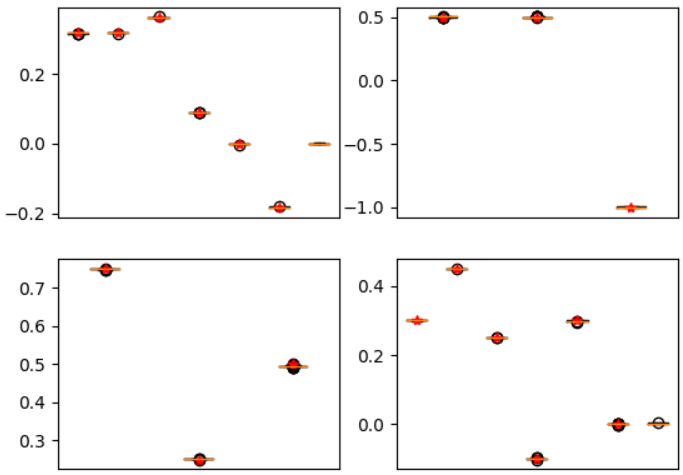


Figure A.8: Distribution of the parameters of the aggregators of 50 HCI networks with C3 aggregators, 1,000 training examples generated by the standard tree model.

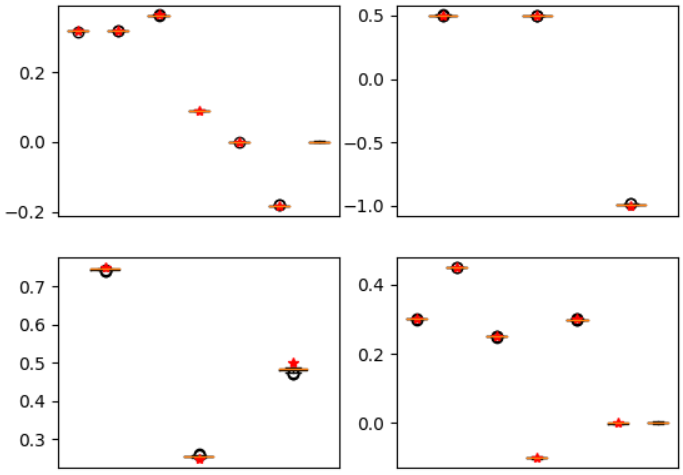


Figure A.9: Distribution of the parameters of the aggregators of 50 HCI networks with CG aggregators, 1,000 training examples generated by the standard tree model.

A.2 Small Datasets

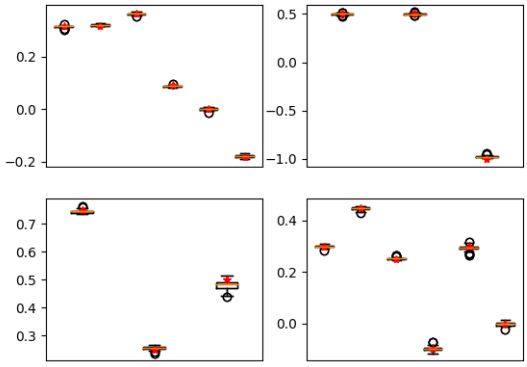


Figure A.10: Distribution of the parameters of the aggregators of an HCI with C2 aggregators, 100 training examples generated by an HCI with 2-additive aggregators. 50 models trained in regression setting.

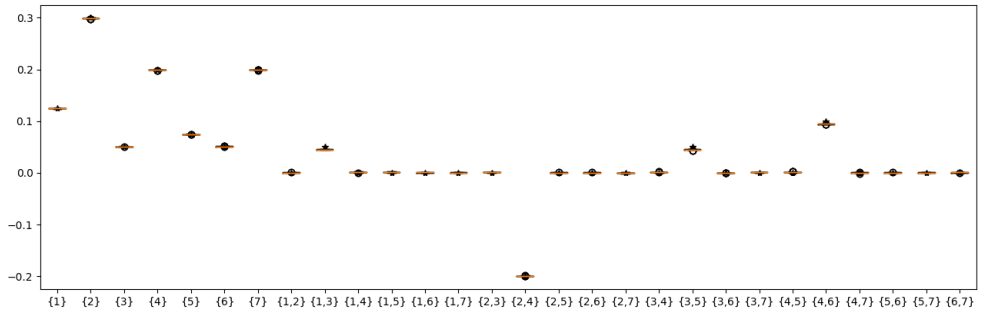


Figure A.11: Distribution of the parameters of the aggregators of a CI with C2 aggregators, 100 training examples generated by a 2-additive CI. 50 models trained in regression setting.

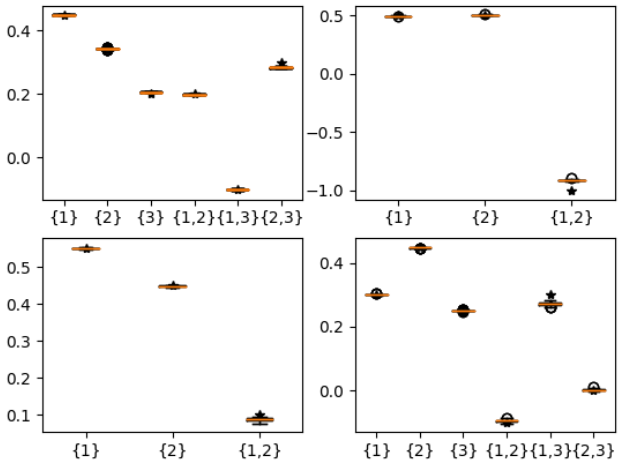


Figure A.12: Distribution of the parameters of the aggregators of a HCI with C2 aggregators, 100 training examples generated by a 2-additive HCI. 50 models trained in regression setting.

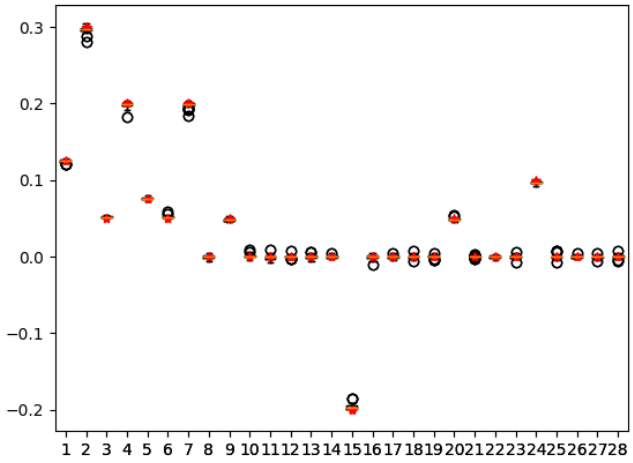


Figure A.13: Distribution of the parameters of the aggregators of a flat CI with C2 aggregators, 100 training examples generated by a flat CI with 2-additive aggregators. 50 models trained in regression setting.

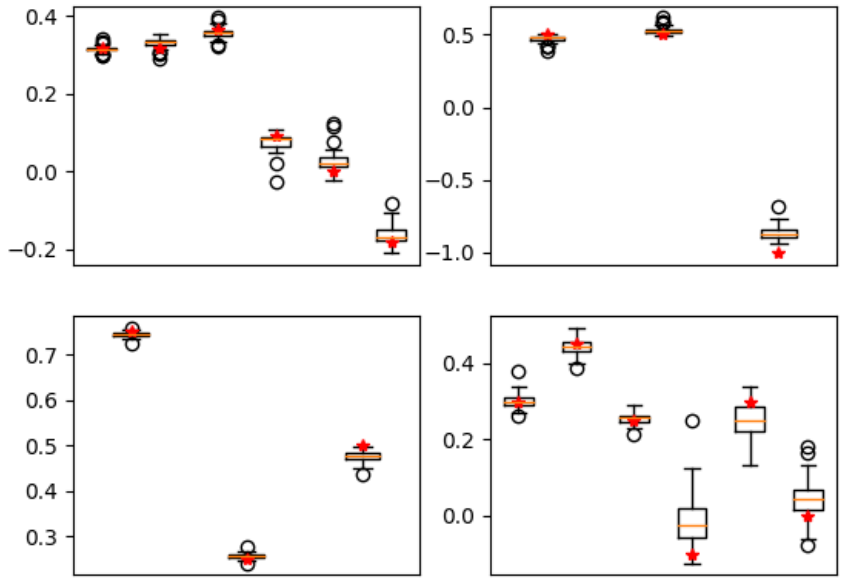


Figure A.14: Plotting the marginal utilities of 50 models trained in regression setting, 30 training examples generated by a UHCI. The aggregators are 2-additive.

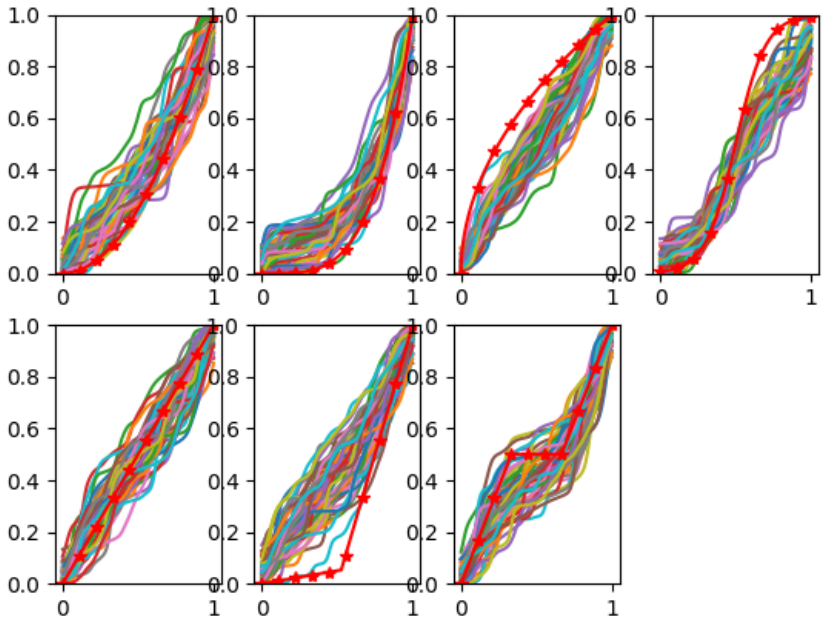


Figure A.15: Plotting the marginal utilities of 50 models trained in regression setting, 30 training examples generated by a UHCI. The aggregators are 2-additive.

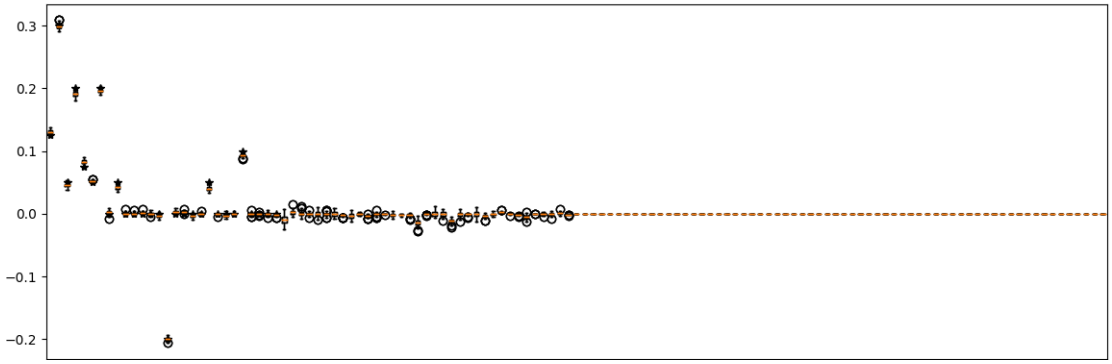


Figure A.16: Distribution of the parameters of the aggregators of a CI with 3-additive aggregators, 300 training examples generated by a flat 2-additive CI. 50 models trained in regression setting.

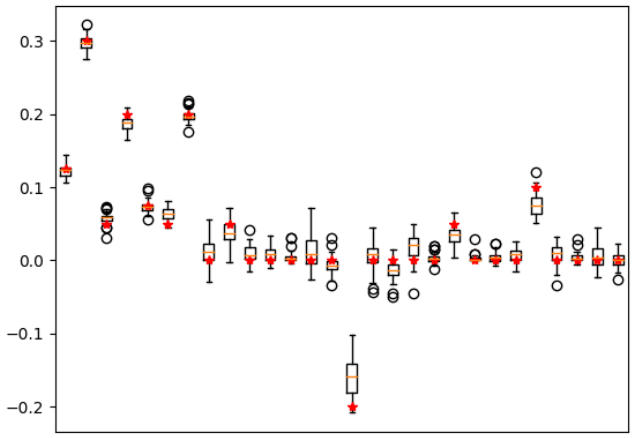


Figure A.17: Learned parameters of 50 2-additive CI models trained on binary classification data generated by a 2-additive CI. The training set was composed of 1000 examples, with a noise level of 0.

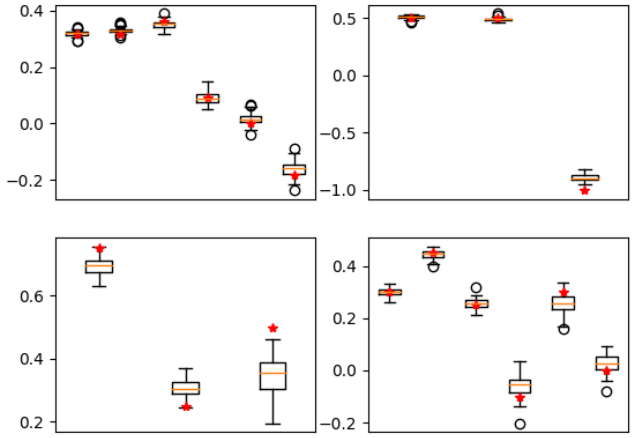


Figure A.18: Learned parameters of 50 2-additive HCI models trained on binary classification data generated by a 2-additive HCI. The training set was composed of 1000 examples, with a noise level of 0.

A.3 Noisy Datasets

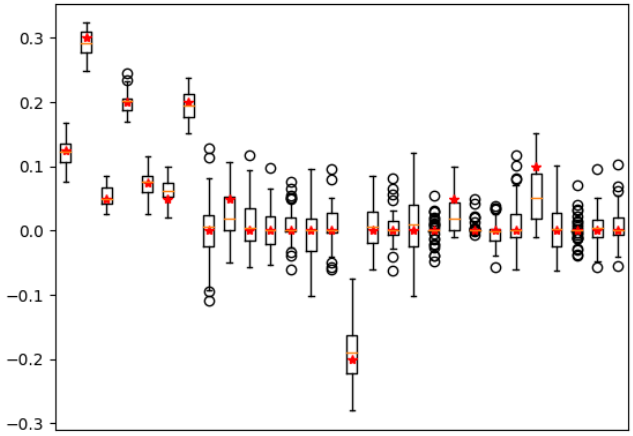


Figure A.19: Learned parameters of 50 2-additive CI models trained on data generated by a 2-additive CI. The training set was composed of 1000 examples, with a gaussian noise with standard deviation 0.1

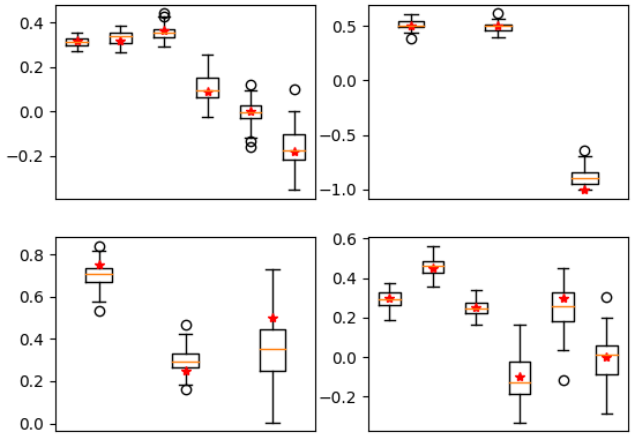


Figure A.20: Learned parameters of 50 2-additive CI models trained on data generated by a 2-additive CI. The training set was composed of 1000 examples, with a gaussian noise with standard deviation 0.1

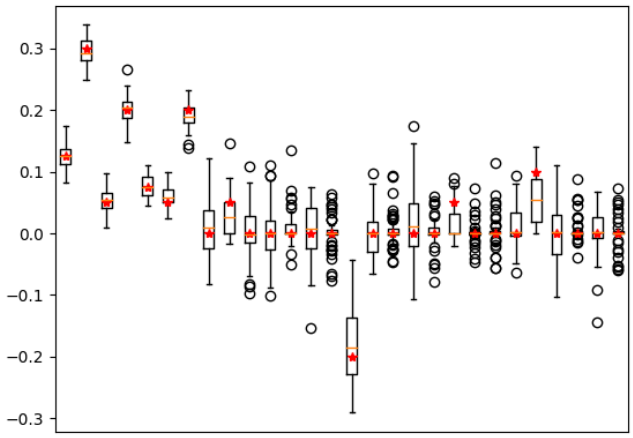


Figure A.21: Learned parameters of 50 2-additive CI models trained on data generated by a 2-additive CI. The training set was composed of 1000 examples, with a gaussian noise with standard deviation 0.2

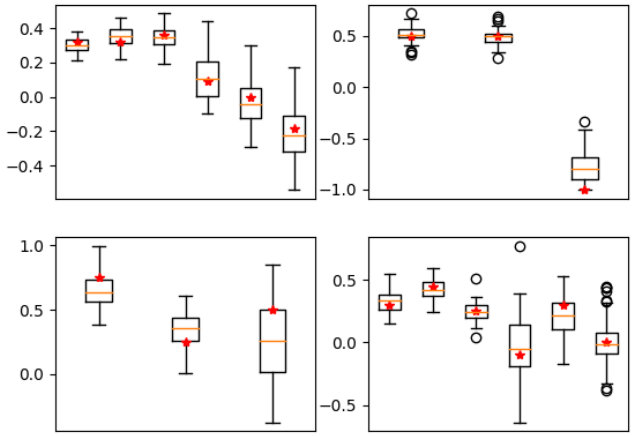


Figure A.22: Learned parameters of 50 2-additive CI models trained on data generated by a 2-additive CI. The training set was composed of 1000 examples, with a gaussian noise with standard deviation 0.2

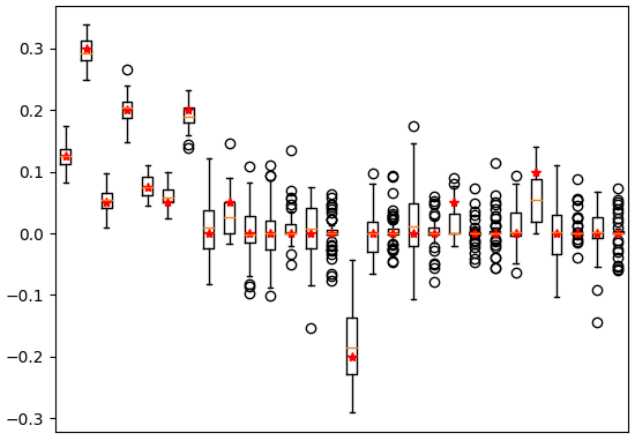


Figure A.23: Learned parameters of 50 2-additive CI models trained on data generated by a 2-additive CI. The training set was composed of 1000 examples, with a gaussian noise with standard deviation 0.2

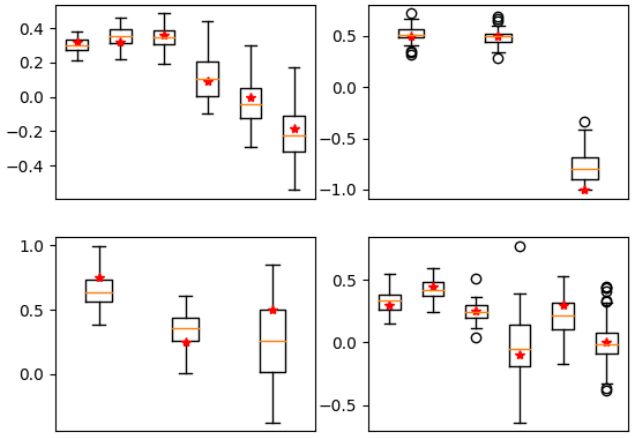


Figure A.24: Learned parameters of 50 2-additive CI models trained on data generated by a 2-additive CI. The training set was composed of 1000 examples, with a gaussian noise with standard deviation 0.2

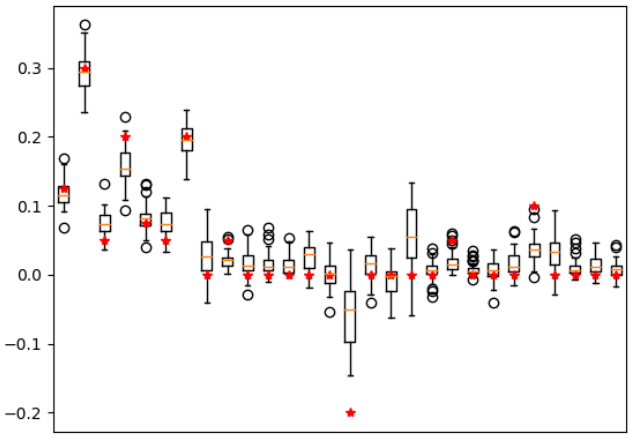


Figure A.25: Learned parameters of 50 2-additive HCI models trained on binary classification data generated by a 2-additive HCI. The training set was composed of 300 examples, with a noise level of 0.0.

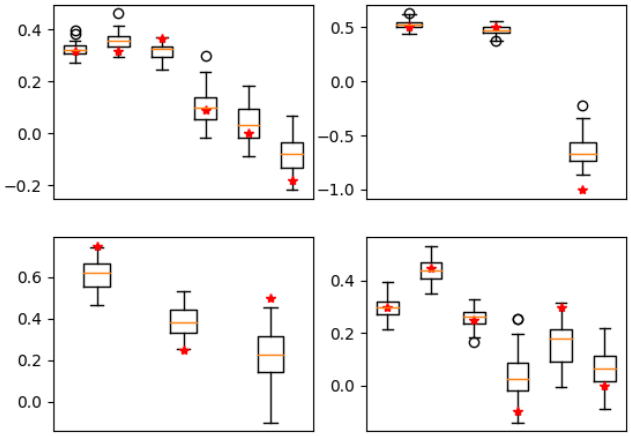


Figure A.26: Learned parameters of 50 2-additive HCI models trained on binary classification data generated by a 2-additive HCI. The training set was composed of 300 examples, with a noise level of 0.0.

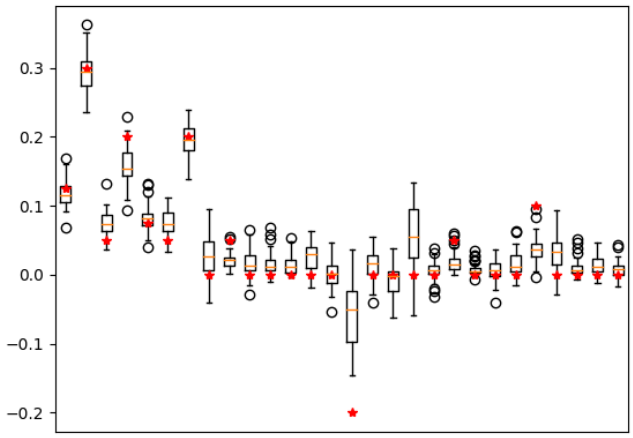


Figure A.27: Learned parameters of 50 2-additive HCI models trained on binary classification data generated by a 2-additive HCI. The training set was composed of 300 examples, with a noise level of 0.05.

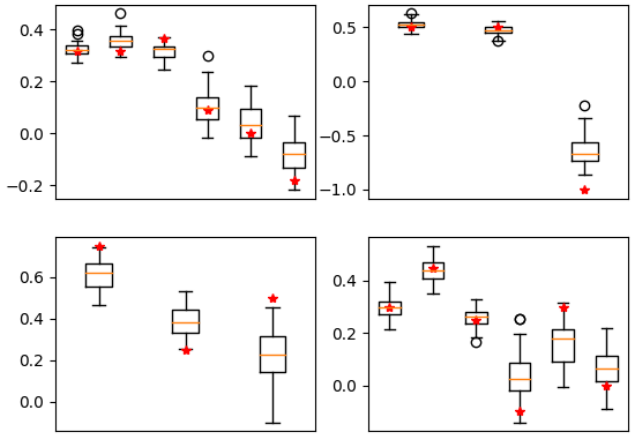


Figure A.28: Learned parameters of 50 2-additive HCI models trained on binary classification data generated by a 2-additive HCI. The training set was composed of 300 examples, with a noise level of 0.05.

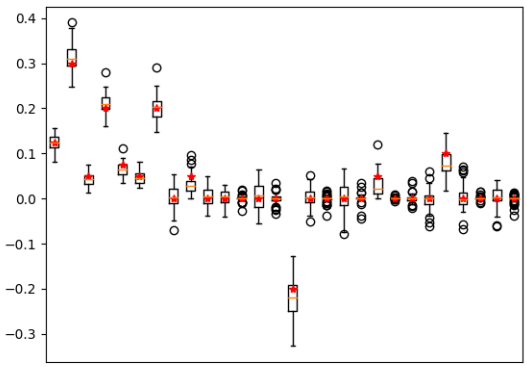


Figure A.29: Learned parameters of 50 2-additive CI models in pairwise preference settings. The data was composed of 300 pairs of alternatives, with the preferred one given each time. A given alternative a was considered prefered to b if $Q(a) > Q(b) + 0.1$, with Q the standard flat model. The red star denote the ground truth parameters.

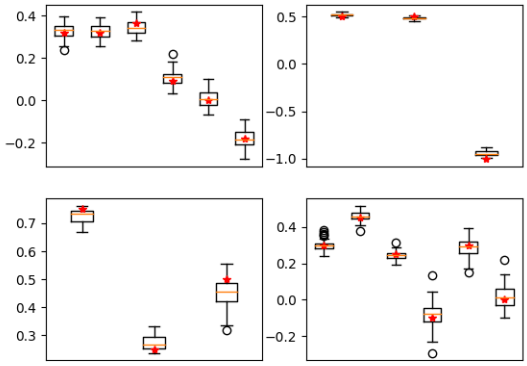


Figure A.30: Learned parameters of 50 2-additive HCI models in pairwise preference settings. The data was composed of 300 pairs of alternatives, with the preferred one given each time. A given alternative a was considered prefered to b if $Q(a) > Q(b) + 0.1$, with Q the standard tree model. The red star denote the ground truth parameters.

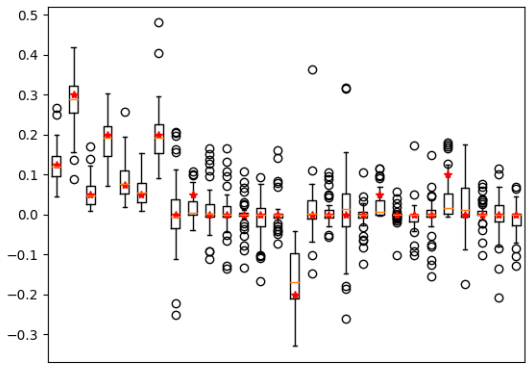


Figure A.31: Learned parameters of 50 2-additive CI models in pairwise preference settings. The data was composed of 300 pairs of alternatives, with the preferred one given each time. A given alternative a was considered prefered to b if $Q(a) > Q(b) + 0.1$, with Q the standard flat model. The red star denote the ground truth parameters. The noise level is at 0.1, i.e. 10% of the training pairs were inverted.

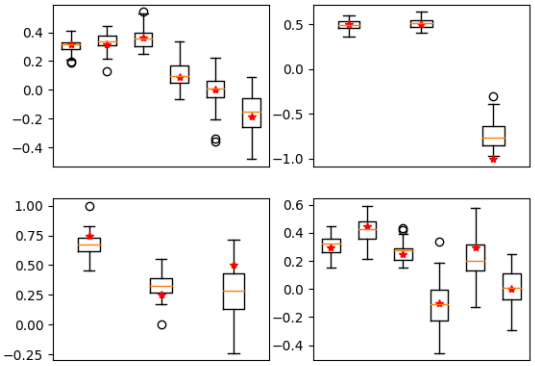


Figure A.32: Learned parameters of 50 2-additive HCI models in pairwise preference settings. The data was composed of 300 pairs of alternatives, with the preferred one given each time. A given alternative a was considered prefered to b if $Q(a) > Q(b) + 0.1$, with Q the standard tree model. The red star denote the ground truth parameters. The noise level is at 0.1, i.e. 10% of the training pairs were inverted.

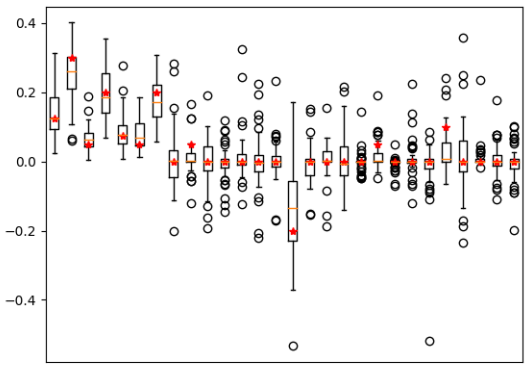


Figure A.33: Learned parameters of 50 2-additive CI models in pairwise preference settings. The data was composed of 300 pairs of alternatives, with the preferred one given each time. A given alternative a was considered prefered to b if $Q(a) > Q(b) + 0.1$, with Q the standard flat model. The red star denote the ground truth parameters. The noise level is at 0.2, i.e. 20% of the training pairs were inverted.

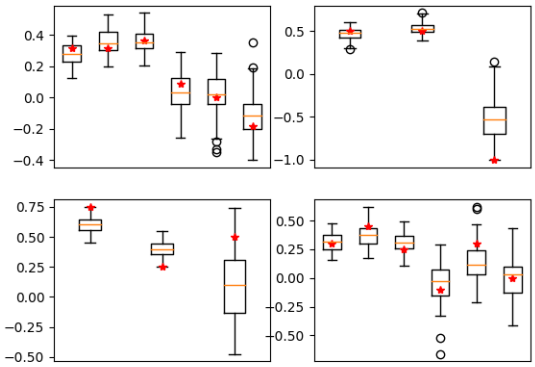


Figure A.34: Learned parameters of 50 2-additive HCI models in pairwise preference settings. The data was composed of 300 pairs of alternatives, with the preferred one given each time. A given alternative a was considered prefered to b if $Q(a) > Q(b) + 0.1$, with Q the standard tree model. The red star denote the ground truth parameters. The noise level is at 0.2, i.e. 20% of the training pairs were inverted.

A.4 Corner Points - Binary Alternatives

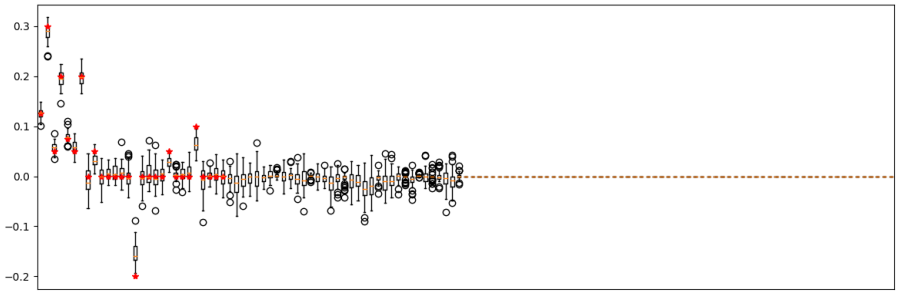


Figure A.35: Distribution of the parameters of the aggregators of an HCI with C3 aggregators, trained on 30 randomly drawn corners of the unit hypercube (different for each trained model) , with the standard flat model as the ground truth.

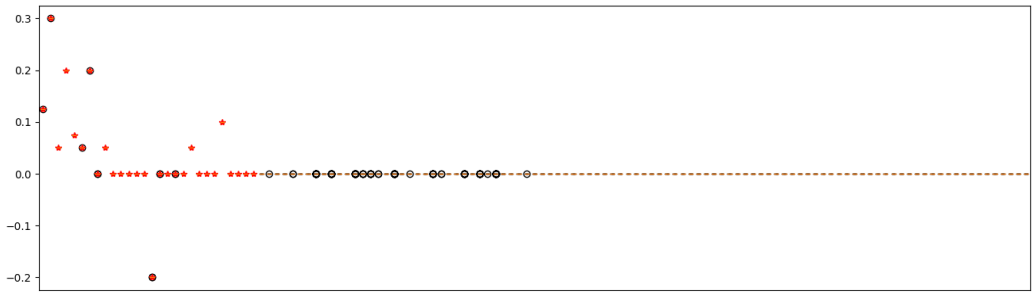


Figure A.36: Distribution of the parameters of the aggregators of an HCI with C3 aggregators, trained on all 128 corners of the unit hypercube, with the standard flat model as the ground truth.

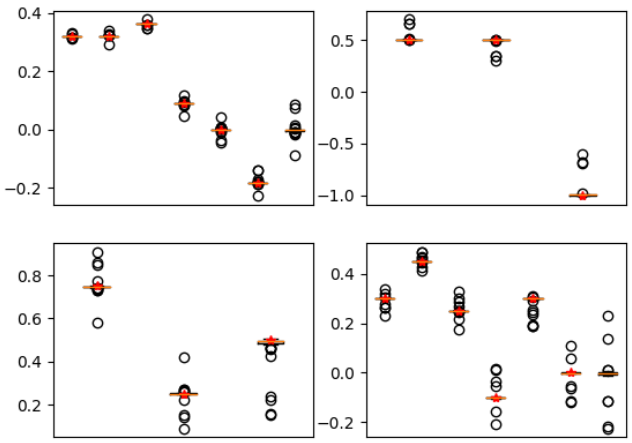


Figure A.37: Distribution of the parameters of the aggregators of an HCI with C3 aggregators, trained on 30 randomly drawn corners of the unit hypercube (different for each trained model) , with the standard tree model as the ground truth.

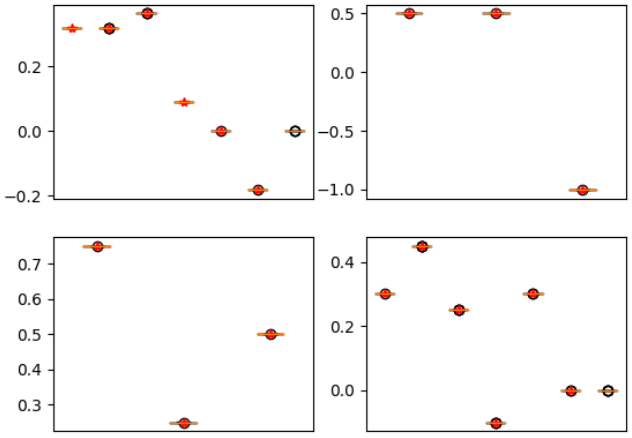


Figure A.38: Distribution of the parameters of the aggregators of an HCI with C3 aggregators, trained on all 128 corners of the unit hypercube, with the standard tree model as the ground truth.

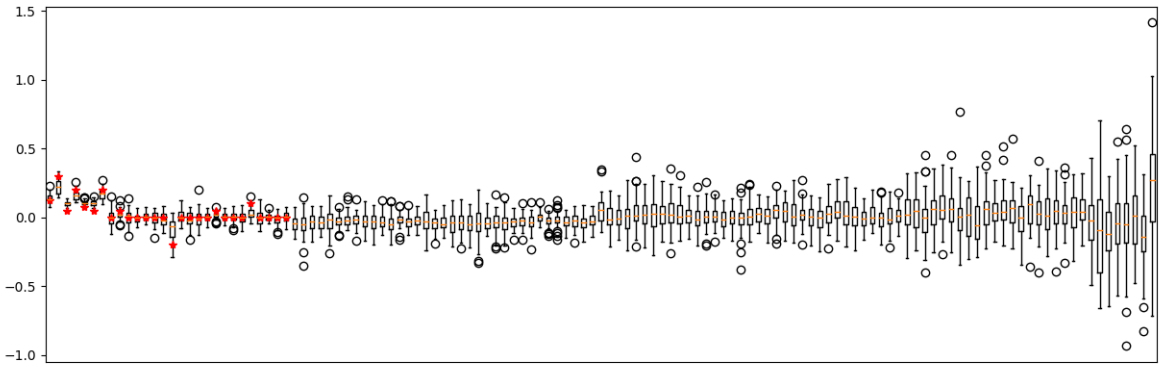


Figure A.39: Distribution of the parameters of the aggregators of an HCI with CG aggregators, trained on 30 randomly drawn corners of the unit hypercube (different for each trained model) , with the standard flat model as the ground truth.

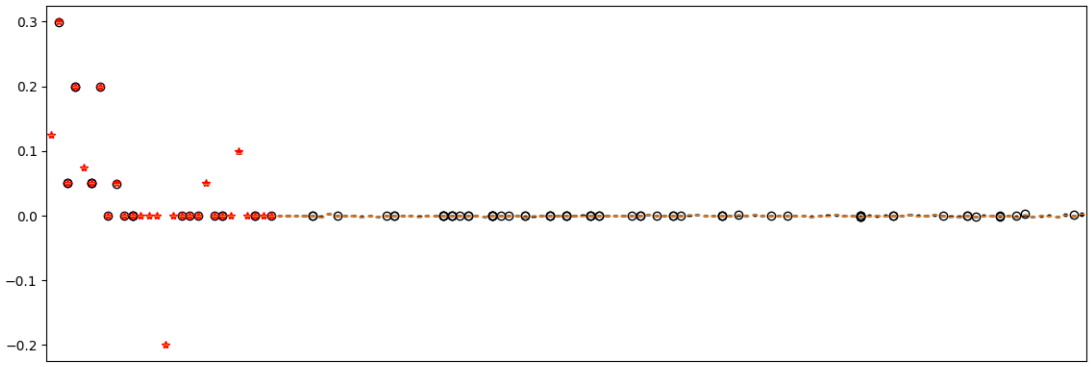


Figure A.40: Distribution of the parameters of the aggregators of an HCI with CG aggregators, trained on all 128 corners of the unit hypercube, with the standard flat model as the ground truth.

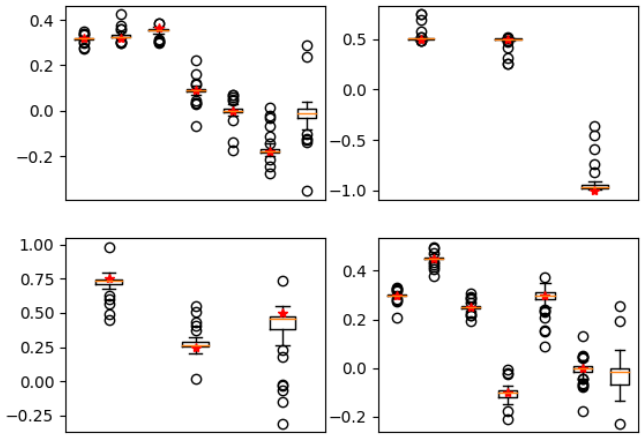


Figure A.41: Distribution of the parameters of the aggregators of an HCI with CG aggregators, trained on 30 randomly drawn corners of the unit hypercube (different for each trained model) , with the standard tree model as the ground truth.

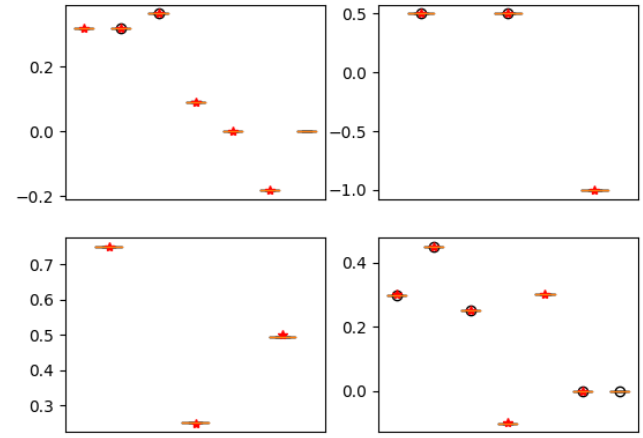


Figure A.42: Distribution of the parameters of the aggregators of an HCI with CG aggregators, trained on all 128 corners of the unit hypercube, with the standard tree model as the ground truth.

A.5 Learning Without a Known Hierarchy

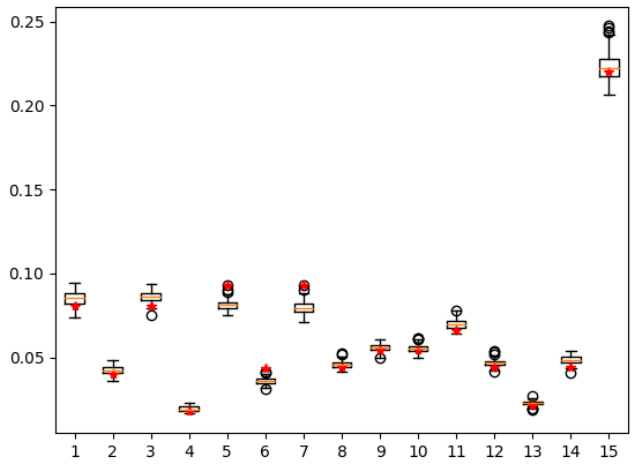


Figure A.43: Extended Shapley values for 50 tree models, each with a randomly generated hierarchy, trained on (different) datasets (1000 examples each, dimension 15) generated by the same randomly generated 2-additive model (random hierarchy and weights).

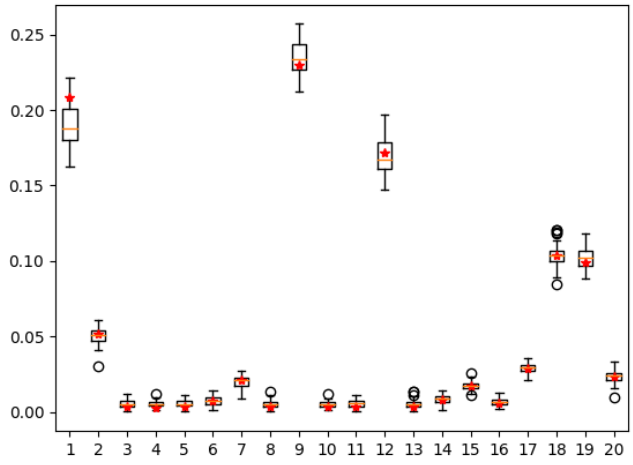


Figure A.44: Extended Shapley values for 50 tree models, each with a randomly generated hierarchy, trained on (different) datasets (1000 examples each, dimension 20) generated by the same randomly generated 2-additive model (random hierarchy and weights).

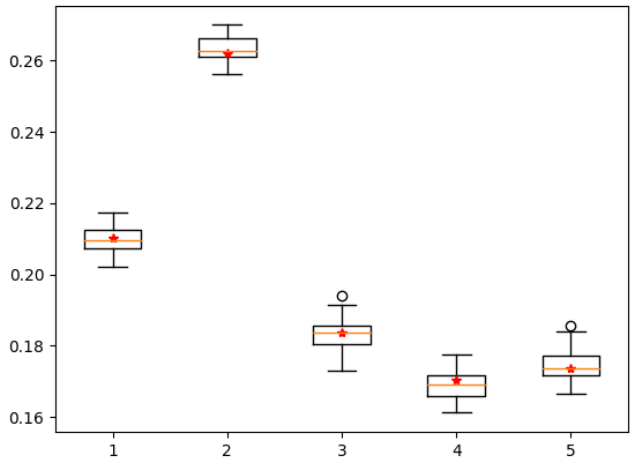


Figure A.45: Extended Shapley values for 50 tree models, each with a randomly generated hierarchy, trained on (different) datasets (1000 examples each, dimension 5) generated by the same randomly generated flat 2-additive model (random weights).

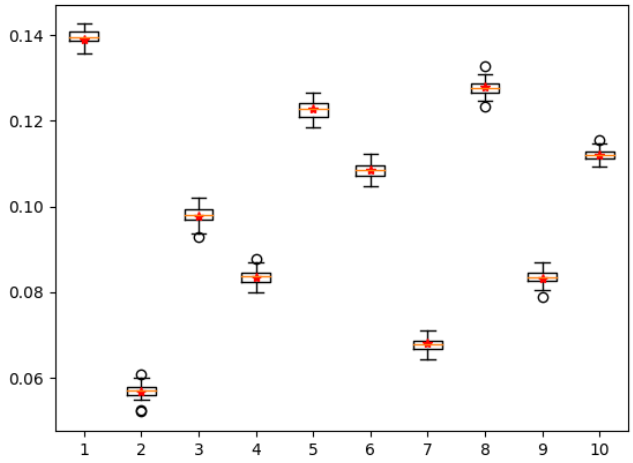


Figure A.46: Extended Shapley values for 50 tree models, each with a randomly generated hierarchy, trained on (different) datasets (1000 examples each, dimension 10) generated by the same randomly generated flat 2-additive model (random weights).

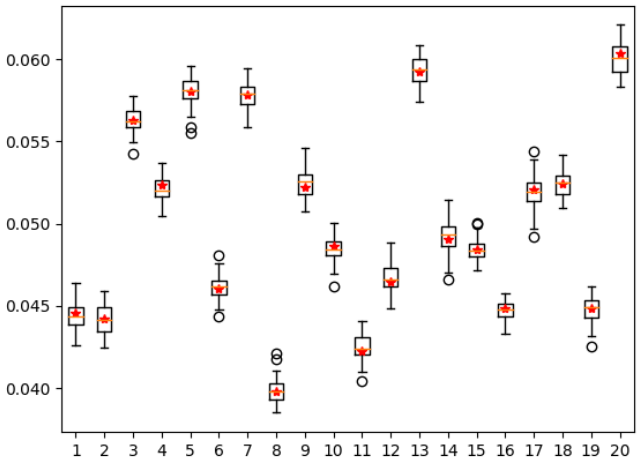


Figure A.47: Extended Shapley values for 50 tree models, each with a randomly generated hierarchy, trained on (different) datasets (1000 examples each, dimension 20) generated by the same randomly generated flat 2-additive model (random weights).

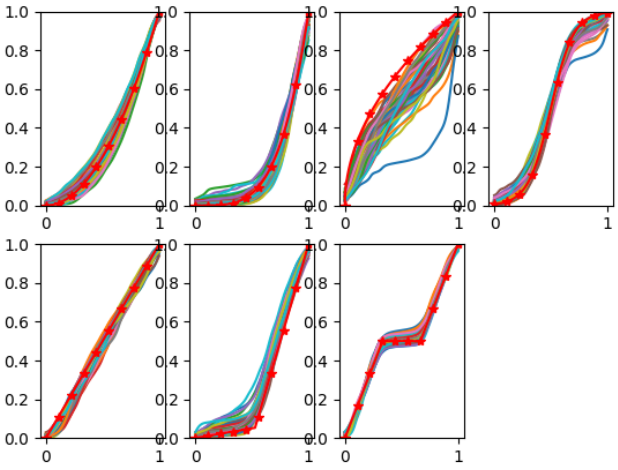


Figure A.48: Marginal utilites of 50 random UHCIs (random hierarchy) trained on data generated by the standard tree model, in regression setting, over 3000 data points.

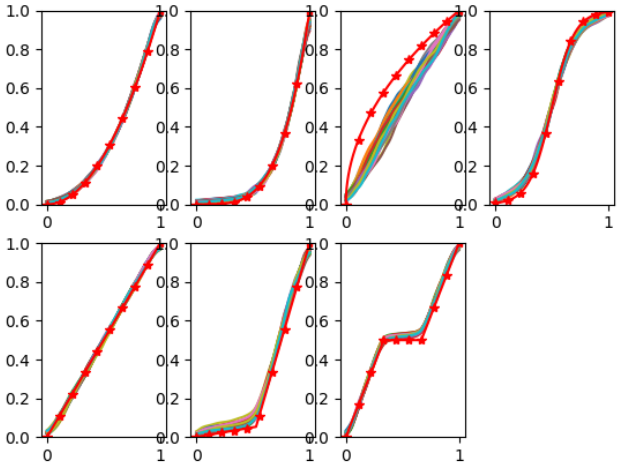


Figure A.49: Marginal utilites of 50 random UCIs (flat hierarchy) trained on data generated by the standard tree model, in regression setting, over 3000 data points.

A.6 Stability of DAG-Winter Values

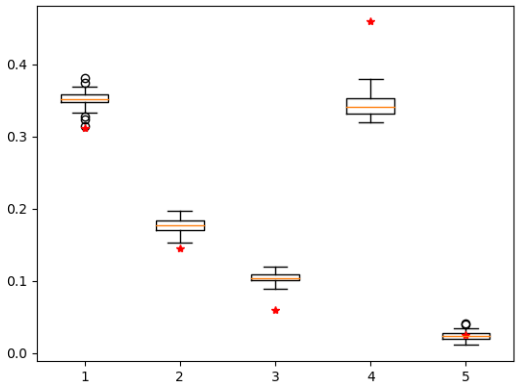


Figure A.50: DAG Winter values for 50 DAG models, trained on (different) datasets (100 examples each, dimension 5) generated by the same randomly generated model (random hierarchy and weights). We see a significant bias between the true and the learned values.

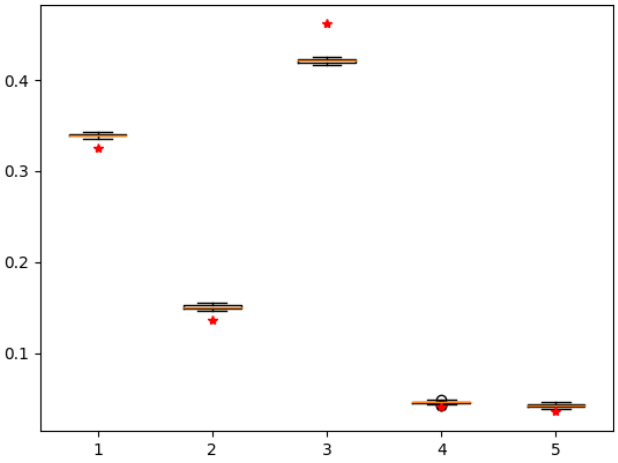


Figure A.51: DAG Winter values for 50 DAG models, trained on (different) datasets (1000 examples each, dimension 5) generated by the same randomly generated model (random hierarchy and weights).

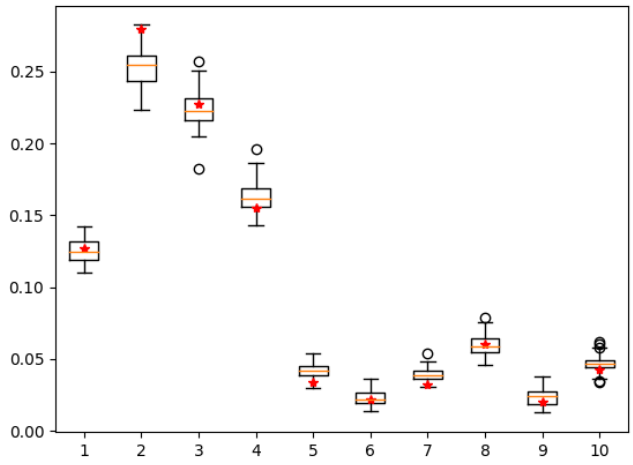


Figure A.52: DAG Winter values for 50 DAG models, trained on (different) datasets (100 examples each, dimension 10) generated by the same randomly generated model (random hierarchy and weights). We see a significant bias between the true and the learned values.

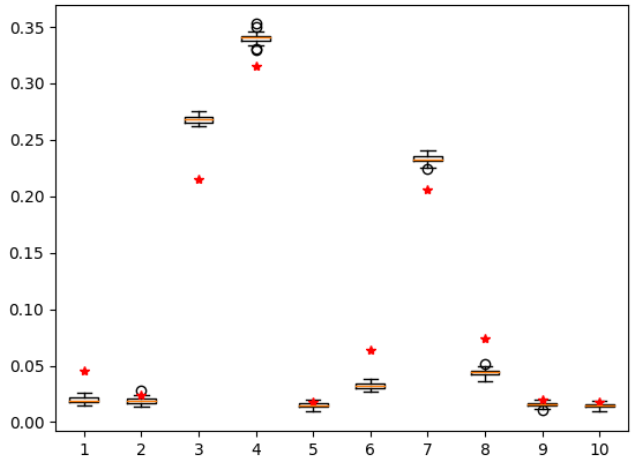


Figure A.53: DAG Winter values for 50 DAG models, trained on (different) datasets (1000 examples each, dimension 10) generated by the same randomly generated model (random hierarchy and weights).

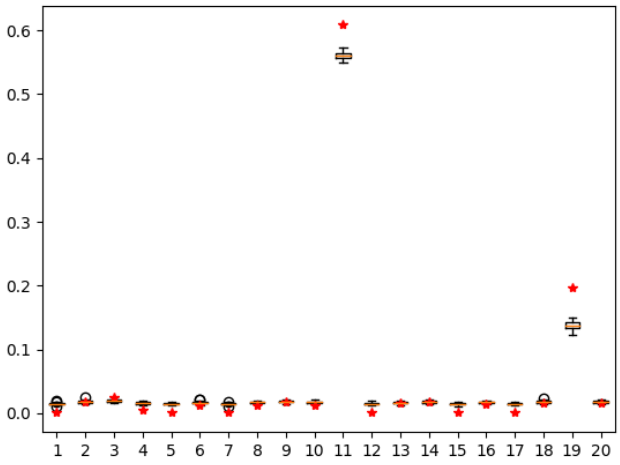


Figure A.54: DAG Winter values for 50 DAG models, trained on (different) datasets (1000 examples each, dimension 20) generated by the same randomly generated model (random hierarchy and weights). We see a significant bias between the true and the learned values.

A.7 Training Time

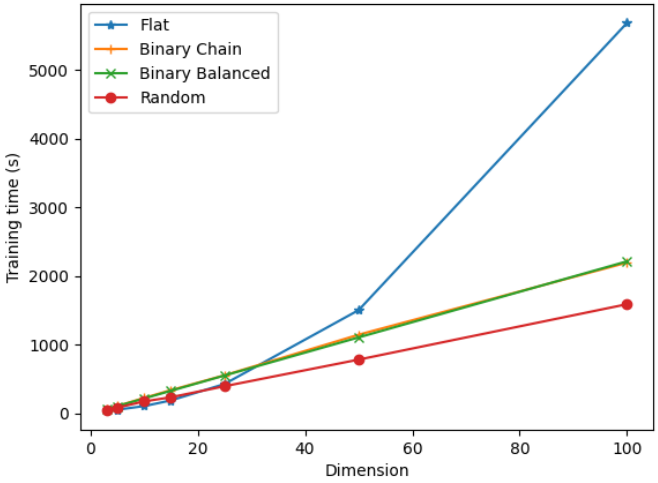


Figure A.55: Training time (s) w.r.t. the dimension of the model. Comparison between flat architectures, binary architectures and randomly generated trees. 2-additive aggregators.

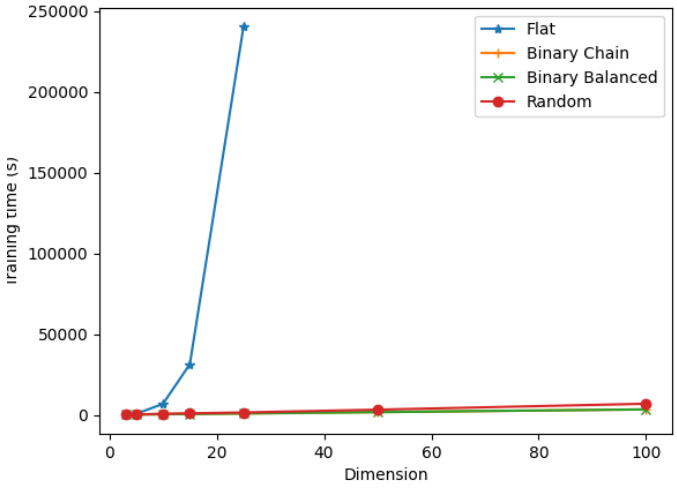


Figure A.56: Training time (s) w.r.t. the dimension of the model. Comparison between flat architectures, binary architectures and randomly generated trees. 3-additive aggregators.

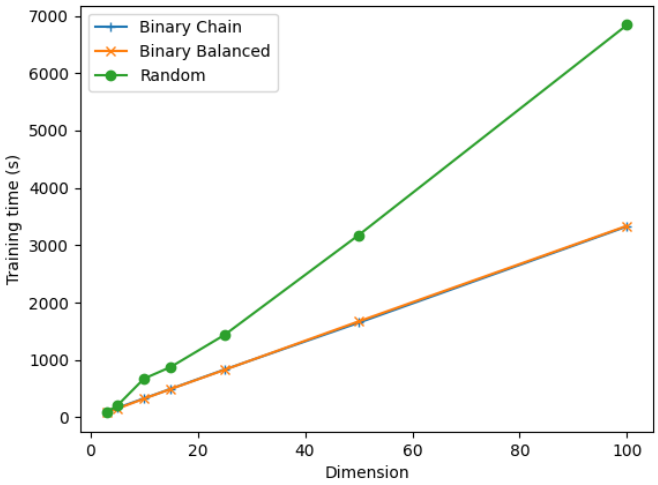


Figure A.57: Same setting as in Figure A.56, without the flat model, to better compare the non-flat models.

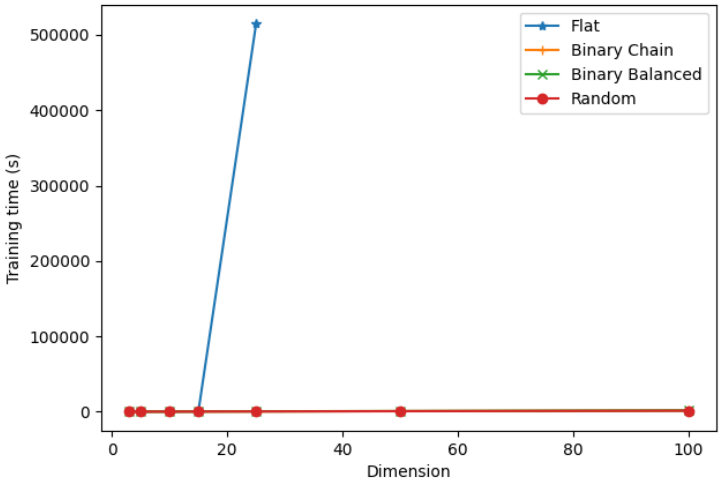


Figure A.58: Training time (s) w.r.t. the dimension of the model. Comparison between flat architectures, binary architectures and randomly generated trees. General aggregators.

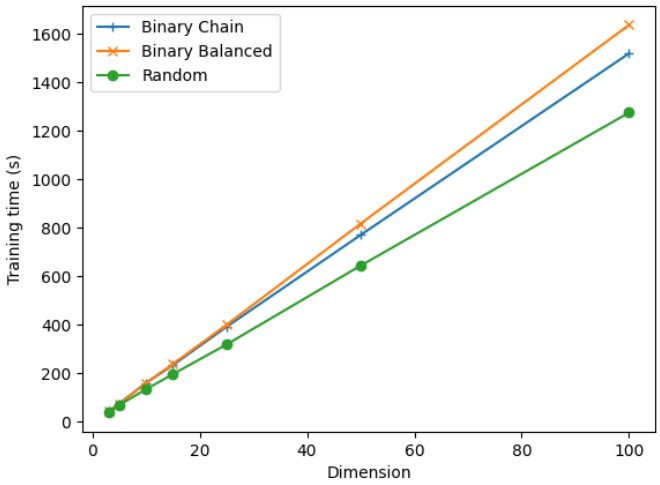


Figure A.59: Same setting as in Figure A.58, without the flat model, to better compare the non-flat models.

APPENDIX B

RÉSUMÉ EN FRANCAIS

Cette thèse a donné lieu à quatre publications:

Neural Representation and Learning of Hierarchical 2-additive Choquet Integrals
R Bresson, J Cohen, E Hüllermeier, C Labreuche, M Sebag; International Joint Conference on Artificial Intelligence (IJCAI-20), Pages 1984-1991

Learning 2-additive Hierarchical Choquet Integrals with non-monotonic utilities
R Bresson, J Cohen, E Hüllermeier, C Labreuche, M Sebag; Proceedings, From Multiple Criteria Decision Aid to Preference Learning, (DA2PL2020)

Evaluating the stability of the Neur-HCI framework
R Bresson, J Cohen, E Hüllermeier, C Labreuche, M Sebag; Actes de la conférence (CAID 2020), Pages 120-128

On the Identifiability of Hierarchical Decision Models
R Bresson, J Cohen, E Hüllermeier, C Labreuche, M Sebag; 18th International Conference on Principles of Knowledge Representation and Reasoning (KR2021), Pages 151-162

Ca manuscrit est organisée comme suit :

B.1 Partie II : Contexte et travaux existants

Dans cette partie, nous présentons les notions de base nécessaires pour établir le contexte de notre travail. Cette partie est divisée en deux chapitres, car cette thèse se situe à la croisée de deux grands domaines. Tout d'abord, dans le chapitre 2, nous introduisons le domaine de l'aide multicritère à la décision (AMCD).

Nous présentons d'abord les notions de base et les motivations du domaine, en particulier la notion d'alternatives définies sur des attributs. Nous détaillons les motivations du domaine ; c'est-à-dire, construire des critères qui représenteront le degré de satisfaction d'une alternative sur un attribut donné, afin d'établir des modèles de décision qui représentent les préférences d'un décideur. De tels modèles sont interprétables et hautement contraints (fiables), ce qui les rend aptes à être utilisés dans des systèmes critiques pour la sécurité. Nous détaillons plusieurs classes de modèles, ainsi que des méthodes de construction de modèles.

Nous nous concentrons en particulier sur les modèles décomposables. Ces modèles ont la propriété intéressante suivante : la satisfaction apportée par un attribut donné ne dépend pas de la valeur des autres attributs. Si cela restreint les comportements possibles du modèle, cela apporte également de l'intelligibilité au modèle, qui peut alors être interprété beaucoup plus facilement.

La famille de modèles sur laquelle nous nous concentrons spécifiquement est l'intégrale de Choquet (IC), une généralisation de la somme pondérée qui permet de prendre en compte les interactions entre les critères (synergie, indépendance et redondance). Nous généralisons à sa version hiérarchique (ICH). L'ICH est un agrégateur hautement interprétable, qui sera utilisé pour calculer un score global, pour une alternative, étant donné ses satisfactions attribut par attribut. Pour cela, il faut que ces dernières soient facilement disponibles, ce qui n'est pas toujours le cas lorsqu'on nous donne les valeurs brutes des attributs. Par conséquent, une ICH peut également être combinée avec ce que l'on appelle des *utilités marginales*, qui sont des fonctions réelles de dimension 1 sur les attributs, et qui permettent de calculer la satisfaction apportée par une certaine valeur de l'attribut donné. une ICH ainsi équipé d'utilités marginales est appelé une ICHU. Nous détaillons ces modèles, car ils sont au cœur des contributions de cette thèse.

Les ICHU ont les avantages des modèles d'AMCD, en ce sens qu'ils sont interprétables et contraints. Néanmoins, ils souffrent également des inconvénients habituels de ce domaine, qui est fortement lié à la recherche opérationnelle. Elle nécessite des informations préférentielles cohérentes, obtenues souvent de manière itérative par une interaction avec un expert du domaine. Ce processus est donc coûteux, et son efficacité est limitée par la source humaine d'information. Enfin, il ne permet pas d'exploiter des données existantes et bruitées, qui pourraient être facilement disponibles dans certains contextes.

Par conséquent, nous nous tournons vers l'apprentissage machine (ML). Dans le chapitre 3, nous introduisons le domaine de l'apprentissage automatique supervisé. Nous présentons les concepts sous-jacents et les méthodes largement utilisées. Nous donnons ensuite un bref apercu d'une famille de modèles appelés réseaux de neurones (NN). En effet, ces modèles ont une structure hiérarchique inhérente, qui est parfaitement adaptée pour représenter les ICHU.

Nous présentons les problèmes récurrents présents dans les ML, en particulier ceux dus à l'incertitude, ou les comportements problématiques, tels que l'overfitting.

Nous introduisons ensuite les notions de base de l'apprentissage des préférences, un domaine qui se concentre sur l'apprentissage de modèles de préférences à partir de données, comme une approche plus statistique et plus orientée vers les données des problèmes de décision que le MCDA aborde de manière orientée vers les contraintes.

Nous développons en particulier des méthodes de ML visant à apprendre des modèles à partir du domaine de la MCDA, mais nous concluons que, si l'IC est un point d'intérêt depuis quelques années, les ICHU n'ont jamais été appris auparavant.

B.2 Partie III : Contribution théorique

La partie III est la première des deux parties consacrées aux contributions de cette thèse. Elle consiste en une preuve d'identifiabilité du modèle ICHU, c'est-à-dire que nous montrons qu'il n'y a qu'une seule paramétrisation possible pour un modèle donné. Cela signifie que, pour un ICHU $\mathcal{F}$ donné qui respecte certaines conditions faibles, alors $\mathcal{F}$ ne peut avoir que :

- une seule hiérarchie ;
- un seul ensemble d'utilités marginales ;
- un seul ensemble de poids pour ses agrégateurs.

Il s'agit d'un résultat très intéressant pour obtenir un modèle fiable. En effet, l'intelligibilité et l'interprétation d'un ICHU résident dans ses paramètres. S'assurer de l'existence d'une seule paramétrisation, c'est s'assurer qu'il ne peut y avoir deux ou plusieurs interprétations contradictoires sur un modèle donné ; ce qui rendrait la confiance dans le modèle impossible en pratique.

Cette preuve est faite en deux étapes, réparties entre les deux chapitres de la partie. Premièrement, le chapitre 4 présente la preuve de l'unicité lorsque le modèle est défini sur une hiérarchie fixe. En d'autres termes, nous supposons

une hiérarchie donnée, et montrons que deux modèles qui sont égaux partout sur l'espace des attributs ont nécessairement les mêmes agrégateurs et les mêmes utilités marginales. Nous procédons principalement en explorant les sommets de l'hypercube d'entrée, ce qui nous permet d'isoler chaque élément et de montrer son unicité.

Ensuite, le chapitre 5 se généralise au cas où la hiérarchie est libre. Nous poursuivons en considérant une autre représentation de l'ICHU. En effet, cette classe de modèles, étant donné des utilités marginales C^1 par morceaux (une hypothèse faible), est elle-même C^1 par morceaux. Notre stratégie consiste donc à étudier les frontières entre chaque région sur laquelle une ICHU est C^1. Ce faisant, nous sommes en mesure de montrer que sous deux hypothèses faibles, ces frontières sont uniques pour un modèle donné, et qu'un seul modèle peut avoir ces frontières, complétant ainsi efficacement la preuve.

L'intérêt supplémentaire de cette preuve est qu'elle constitue un résultat encourageant pour l'apprentissage des ICHU à l'aide de méthodes statistiques.

B.3 Partie IV : Contribution technique

Dans cette partie, nous présentons la principale contribution technique de cette thèse. Cette contribution est le framework NEUR-HCI, qui est un type de modules neuronaux spécifiques (c'est-à-dire de petits réseaux neuronaux), chacun représentant une partie spécifique (agrégateur de CI, ou utilité marginale) d'un modèle d'ICHU.

Le chapitre 6 présente les modules implémentant les utilités marginales. Nous traitons 4 types d'utilités marginales :

- décroissante ;
- croissante ;
- simple plateau (croissante puis décroissante) ;
- simple vallée (décroissante puis croissante).

Ces modules sont construits de manière à pouvoir représenter toute (et seulement) utilité marginale de leur type respectif. Leur conception garantit que toutes les contraintes requises par le modèle, en termes de monotonie et de normalisation, sont respectées, de sorte que le modèle reste valide à tout moment.

Nous présentons également un module appelé *selecteur*, qui permet de choisir, parmi les quatre types présentés ci-dessus, celui qui est le plus adapté aux données d'apprentissage.

Ensuite, le chapitre 7 présente les architectures neuronales qui représentent les fonctions agrégatrices basées sur CI. Nous présentons trois types d'intégrales de Choquet :

- 2-additif (qui permet les interactions de deux critères au maximum à la fois)
- un sous-ensemble des intégrales de Choquet 3-additives (qui permet des interactions de jusqu'à 3 critères à la fois)
- général (toute intégrale de Choquet de la dimension donnée)

Tout comme pour les modules d'utilité marginale, nous assurons par conception la validité de l'IC représentée par le module, en termes de contraintes formelles.

Enfin, le chapitre 8 explique comment nous formons un réseau neuronal construit à partir des modules introduits dans les deux chapitres précédents. Les paramètres de formation sont les suivants

- **Régression :** le modèle recoit un ensemble d'alternatives, ainsi que leurs scores attendus, et apprend à prédire les scores des nouvelles alternatives.
- **Classification :** Le modèle recoit un ensemble d'alternatives et leur classe de préférence (très mauvais, mauvais,..., très bon), et apprend à classer les nouvelles alternatives.
- **Apprentissage des préférences par paires :** Le modèle recoit des paires d'alternatives, ainsi que l'information indiquant laquelle des deux alternatives est préférée à l'autre. Il apprend ensuite à prédire de nouvelles préférences.

Nous avons également présenté un théorème de validité, montrant que tout réseau NEUR-HCI est une ICHU valide, et que toute ICHU avec les mêmes types d'utilités marginales et de contraintes d'additivité que nos modules peut être représentée par un réseau NEUR-HCI.

Il est à noter que cette approche a été validée sur une application interne de Thales.

B.4 Partie V : Résultats expérimentaux

Dans cette section, nous testons la performance, la robustesse et la stabilité des modèles présentés dans la partie IV. Nous le faisons sur des données réelles et générées artificiellement. Nous montrons que non seulement les modèles NEUR-HCI peuvent apprendre efficacement un modèle qui s'ajuste bien aux données,

mais qu'ils s'avèrent également stables, c'est-à-dire qu'avec suffisamment de données, les modèles formés sur des données similaires auront des paramètres et une interprétation similaires. Cela est très important pour faire confiance au modèle, et donc pour les applications critiques en matière de sécurité.

B.5 Partie VI : Perspectives et conclusions

Nous présentons enfin les perspectives de travaux futurs, qui pourraient être réalisés pour itérer sur les travaux présentés dans cette thèse. Il s'agit notamment d'extensions des modèles appris, ainsi que d'éventuels travaux théoriques visant à développer de nouveaux indicateurs pour interpréter les modèles MCDA.

BIBLIOGRAPHY

[Abdar et al., 2020] Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., Fieguth, P. W., Cao, X., Khosravi, A., Acharya, U. R., Makarenkov, V., and Nahavandi, S. (2020). A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *CoRR*, abs/2011.06225.

[Abdellaoui and Gonzales, 2013] Abdellaoui, M. and Gonzales, C. (2013). Multi-attribute utility theory. page 36.

[Alavi et al., 2009] Alavi, S. H., Jassbi, J., Serra, P. J. A., and Ribeiro, R. A. (2009). Defining fuzzy measures: A comparative study with genetic and gradient descent algorithms. In Machado, J. A. T., Pátkai, B., and Rudas, I. J., editors, *Intelligent Engineering Systems and Computational Cybernetics*, pages 427–437. Springer Netherlands.

[Amini, 2015] Amini, M.-R. (2015). *Machine Learning*. Eyrolles.

[Angilella et al., 2015] Angilella, S., Corrente, S., and Greco, S. (2015). Stochastic multiobjective acceptability analysis for the choquet integral preference model and the scale construction problem. *European Journal of Operational Research*, 240(1):172–182.

[Angilella et al., 2013] Angilella, S., Corrente, S., Greco, S., and Roman, S. (2013). Multiple criteria hierarchy process for the choquet integral. volume 7811.

[Angilella et al., 2010] Angilella, S., Greco, S., and Matarazzo, B. (2010). Non-additive robust ordinal regression: A multiple criteria decision model based on the choquet integral. *European Journal of Operational Research*, 201(1):277–288.

[Audemard et al., 2021] Audemard, G., Bellart, S., Bounia, L., Koriche, F., Lagniez, J., and Marquis, P. (2021). On the computational intelligibility of boolean classifiers. *CoRR*, abs/2104.06172.

[Bana e Costa and Vansnick, 1999] Bana e Costa, C. and Vansnick, J.-C. (1999). The MACBETH approach: basic ideas, software, and an application. 4.

[Bastani et al., 2016] Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A. V., and Criminisi, A. (2016). Measuring neural net robustness with constraints. *CoRR*, abs/1605.07262.

[Beliakov, 2018] Beliakov, G. (2018). Comparing apples and oranges: The weighted owa function. *International Journal of Intelligent Systems*, 33(5):1089–1108.

[Beliakov and Divakov, 2021] Beliakov, G. and Divakov, D. (2021). Aggregation with dependencies: Capacities and fuzzy integrals. *Fuzzy Sets and Systems*.

[Beliakov et al., 2021] Beliakov, G., Gagolewski, M., and James, S. (2021). Hierarchical data fusion processes involving the möbius representation of capacities. *Fuzzy Sets and Systems*.

[Beliakov and James, 2021] Beliakov, G. and James, S. (2021). Choquet integral optimisation with constraints and the buoyancy property for fuzzy measures. *Information Sciences*, 578:22–36.

[Beliakov and Wu, 2019] Beliakov, G. and Wu, J.-Z. (2019). Learning fuzzy measures from data: Simplifications and optimisation strategies. *Information Sciences*, 494:100–113.

[Beliakov and Wu, 2021] Beliakov, G. and Wu, J.-Z. (2021). Learning k-maxitive fuzzy measures from data by mixed integer programming. *Fuzzy Sets and Systems*, 412:41–52. Fuzzy Measures and Integrals.

[Benabbou et al., 2016a] Benabbou, N., Di Sabatino Di Diodoro, S., Perny, P., and Viappiani, P. (2016a). Incremental preference elicitation in multi-attribute domains for choice and ranking with the borda count. In Schockaert, S. and Senellart, P., editors, *Scalable Uncertainty Management*, volume 9858, pages 81–95. Springer International Publishing.

[Benabbou and Perny, 2017] Benabbou, N. and Perny, P. (2017). Adaptive elicitation of preferences under uncertainty in sequential decision making problems. In *The 26th International Joint Conference on Artificial Intelligence*.

[Benabbou et al., 2016b] Benabbou, N., Perny, P., and Viappiani, P. (2016b). A regret-based preference elicitation approach for sorting with multicriteria reference profiles. *DA2PL'16*.

[Benabbou et al., 2017] Benabbou, N., Perny, P., and Viappiani, P. (2017). Incremental elicitation of choquet capacities for multicriteria choice, ranking and sorting problems. *Artificial Intelligence*, 246:152–180.

[Benayoun et al., 1966] Benayoun, R., Roy, B., and Sussman, B. (1966). ELECTRE: une méthode pour guider le choix en présence des points de vue multiples.

[Bethune et al., 2021] Bethune, L., Gonz'alez-Sanz, A., Mamalet, F., and Serrurier, M. (2021). The many faces of 1-lipschitz neural networks. *ArXiv*, abs/2104.05097.

[Bishop, 2007] Bishop, C. M. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition.

[Boser et al., 1992] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In Haussler, D., editor, *Proceedings of the 5th Annual Workshop on Computational Learning Theory (COLT'92)*, pages 144–152, Pittsburgh, PA, USA. ACM Press.

[Bourdache and Perny, 2019] Bourdache, N. and Perny, P. (2019). Active preference learning based on generalized gini functions: Application to the multiagent knapsack problem. In *Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*.

[Bourdache et al., 2019] Bourdache, N., Perny, P., and Spanjaard, O. (2019). Incremental elicitation of rank-dependent aggregation functions based on bayesian linear regression. In *IJCAI-19 - Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 2023–2029. International Joint Conferences on Artificial Intelligence Organization.

[Bous and Pirlot, 2013] Bous, G. and Pirlot, M. (2013). Learning multicriteria utility functions with random utility models. In Perny, P., Pirlot, M., and Tsoukiàs, A., editors, *Algorithmic Decision Theory*, volume 8176, pages 101–115. Springer Berlin Heidelberg.

[Bouyssou et al., 2006] Bouyssou, D., Marchant, T., Pirlot, M., Tsoukiàs, A., and Vincke, P. (2006). *Evaluation and Decision Models with Multiple Criteria: Stepping Stones for the Analyst*, volume 86.

[Brans and Vincke, 1985] Brans, J. P. and Vincke, P. (1985). A preference ranking organisation method: (the PROMETHEE method for multiple criteria decision-making). *Management Science*, 31(6):647–656.

[Bresson et al., 2020a] Bresson, R., Cohen, J., Hüllermeier, E., Labreuche, C., and Sebag, M. (2020a). Evaluating the stability of the neur-hci framework. In *CAID 2020 - Second Conference on Artificial Intelligence for Defence*, pages 120–126.

[Bresson et al., 2020b] Bresson, R., Cohen, J., Hüllermeier, E., Labreuche, C., and Sebag, M. (2020b). Learning 2-additive hierarchical choquet integrals with non-monotonic utilities.

[Bresson et al., 2020c] Bresson, R., Cohen, J., Hüllermeier, E., Labreuche, C., and Sebag, M. (2020c). Neural representation and learning of hierarchical 2-additive choquet integrals. In Bessiere, C., editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1984–1991. ijcai.org.

[Bresson et al., 2021] Bresson, R., Cohen, J., Hüllermeier, E., Labreuche, C., and Sebag, M. (2021). On the identifiability of hierarchical decision models. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR2021*.

[Bromley et al., 1993] Bromley, J., Guyon, I., Lecun, Y., Säckinger, E., and Shah, R. (1993). Signature verification using a siamese time delay neural network. *Int. J. of Pattern Recognition and AI*, 7:737–744.

[Bunel et al., 2017a] Bunel, R., Turkaslan, I., Torr, P. H. S., Kohli, P., and Kumar, M. P. (2017a). Piecewise linear neural network verification: A comparative study. *CoRR*, abs/1711.00455.

[Bunel et al., 2017b] Bunel, R., Turkaslan, I., Torr, P. H. S., Kohli, P., and Kumar, M. P. (2017b). A unified view of piecewise linear neural network verification. *arXiv:1711.00455 [cs]*.

[Burges et al., 2005] Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. (2005). Learning to rank using gradient descent. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 89–96. ACM. event-place: Bonn, Germany.

[Burges et al., 2006] Burges, C. J. C., Ragno, R., and Le, Q. V. (2006). Learning to rank with nonsmooth cost functions. In Schölkopf, B., Platt, J. C., and Hofmann, T., editors, *Neural Information Processing Systems NIPS*, pages 193–200. MIT Press.

[Bärmann et al., 2017] Bärmann, A., Pokutta, S., and Schneider, O. (2017). Emulating the expert: Inverse optimization through online learning. In *International Conference on Machine Learning*, pages 400–410.

[Cailloux and Destercke, 2017] Cailloux, O. and Destercke, S. (2017). Reasons and means to model preferences as incomplete. *arXiv:1801.01657 [cs]*, 10564:17–30.

[Chen and Huang, 2019] Chen, C.-Y. and Huang, J.-J. (2019). Forming a hierarchical choquet integral with a ga-based heuristic least square method. *Mathematics*, 7(12).

[Choquet, 1954] Choquet, G. (1954). Theory of capacities. *Annales de l'Institut Fourier*, 5:131–295.

[Choromanska et al., 2015] Choromanska, A., Henaff, M., Mathieu, M., Ben Arous, G., and LeCun, Y. (2015). The Loss Surfaces of Multilayer Networks. In Lebanon, G. and Vishwanathan, S. V. N., editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 192–204, San Diego, California, USA. PMLR.

[Cohen et al., 1999] Cohen, W. W., Schapire, R. E., and Singer, Y. (1999). Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270.

[Cox, 1958] Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, 20(2):215–242.

[Daniels and Velikova, 2010] Daniels, H. and Velikova, M. (2010). Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 21(6):906–917.

[Danila, 1986] Danila, N. (1986). Roy b. : Méthodologie multicritère d'aide à la décision.

[Doquet and Sebag, 2020] Doquet, G. and Sebag, M. (2020). pages 343–358.

[Dubois and Prade, 2015] Dubois, D. and Prade, H. (2015). Possibility theory and its applications: Where do we stand? In Kacprzyk, J. and Pedrycz, W., editors, *Springer Handbook of Computational Intelligence*, Springer Handbooks, pages 31–60. Springer Berlin Heidelberg.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[Ehlers, 2017] Ehlers, R. (2017). Formal verification of piece-wise linear feed-forward neural networks. *CoRR*, abs/1705.01320.

[Fallah Tehrani et al., 2011] Fallah Tehrani, A., Cheng, W., and Hüllermeier, E. (2011). Choquistic Regression: Generalizing Logistic Regression using the Choquet Integral. *EUSFLAT-11*.

[Fallah Tehrani et al., 2014] Fallah Tehrani, A., Labreuche, C., and Hüllermeier, E. (2014). Choquistic utilitaristic regression.

[Fallah Tehrani et al., 2012] Fallah Tehrani, A., Cheng, W., Dembczyński, K., and Hüllermeier, E. (2012). Learning monotone nonlinear models using the choquet integral. *Machine Learning*, 89(1):183–211.

[Figueira et al., 2008] Figueira, J., Greco, S., Mousseau, V., and Roman, S. (2008). Interactive multiobjective optimization using a set of additive value functions. pages 97–119.

[Figueira et al., 2009] Figueira, J., Greco, S., and Roman, S. (2009). Building a set additive value functions representing a reference preorder and intensities of preference: Grip method. *European Journal of Operational Research*, 195:460–486.

[Figueira et al., 2016] Figueira, J., Mousseau, V., and Roy, B. (2016). ELECTRE methods. 233:155–185.

[Fishburn, 1967a] Fishburn, P. C. (1967a). Additive utilities with finite sets: Applications in the management sciences. *Naval Research Logistics Quarterly*, 14(1):1–13.

[Fishburn, 1967b] Fishburn, P. C. (1967b). Interdependence and additivity in multivariate, unidimensional expected utility theory. *International Economic Review*, 8(3):335–342.

[Fishburn, 1970] Fishburn, P. C. (1970). *Utility theory for decision making*. Wiley.

[Fürnkranz and Hüllermeier, 2003] Fürnkranz, J. and Hüllermeier, E. (2003). Pairwise preference learning and ranking. In Lavra**v**c, N., Gamberger, D., Blockeel, H., and Todorovski, L., editors, *Machine Learning: ECML 2003*, Lecture Notes in Computer Science, pages 145–156. Springer Berlin Heidelberg.

[Fürnkranz and Hüllermeier, 2011] Fürnkranz, J. and Hüllermeier, E. (2011). Preference learning and ranking by pairwise comparison. In Fürnkranz, J. and Hüllermeier, E., editors, *Preference Learning*, pages 65–82. Springer Berlin Heidelberg.

[Ganin et al., 2016] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks.

[Gilbert et al., 2017] Gilbert, H., Benabbou, N., Perny, P., Spanjaard, O., and Viappiani, P. (2017). Incremental decision making under risk with the weighted expected utility model. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 4588–4594. International Joint Conferences on Artificial Intelligence Organization.

[Goodfellow et al., 2014] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv:1412.6572 [cs, stat]*.

[Gopinath et al., 2017] Gopinath, D., Katz, G., Pasareanu, C. S., and Barrett, C. (2017). DeepSafe: A data-driven approach for checking adversarial robustness in neural networks. *CoRR*, abs/1710.00486.

[Goujon, 2018] Goujon, B. (2018). Preference learning for object ranking and classification with fixed-point algorithm. *DA2PL'2018*.

[Grabisch, 1995] Grabisch, M. (1995). A new algorithm for identifying fuzzy measures and its application to pattern recognition. In *FUZZ-IEEE-95*, pages 145–150.

[Grabisch, 1996] Grabisch, M. (1996). The application of fuzzy integrals in multicriteria decision making. *European Journal of Operational Research*, 89(3):445–456.

[Grabisch, 1997a] Grabisch, M. (1997a). Alternative representations of discrete fuzzy measures for decision making. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 5:587–608.

[Grabisch, 1997b] Grabisch, M. (1997b). K-order additive discrete fuzzy measures and their representation. *Fuzzy Sets and Systems*, 92:167–189.

[Grabisch, 1999] Grabisch, M. (1999). The interaction and möbius representations of fuzzy measures on finite spaces, -additive measures: A survey. page 24.

[Grabisch, 2006] Grabisch, M. (2006). L'utilisation de l'intégrale de choquet en aide multicritère à la décision (french). *European Working Group "Multiple Criteria Decision Aiding, Series n\textsuperscripto3"*, (14).

[Grabisch, 2016] Grabisch, M. (2016). *Set Functions, Games and Capacities in Decision Making*.

[Grabisch et al., 2008] Grabisch, M., Kojadinovic, I., and Meyer, P. (2008). A review of methods for capacity identification in choquet integral based multi-attribute utility theory. *European Journal of Operational Research*, 186(2):766–785.

[Grabisch and Labreuche, 2004] Grabisch, M. and Labreuche, C. (2004). Fuzzy measures and integrals in MCDA. In *Multiple Criteria Decision Analysis*, pages 563–608. Kluwer Academic Publishers.

[Grabisch and Labreuche, 2008] Grabisch, M. and Labreuche, C. (2008). A decade of application of the choquet and sugeno integrals in multi-criteria decision aid. *Annals of Operations Research*, 175.

[Grabisch and Labreuche, 2015] Grabisch, M. and Labreuche, C. (2015). On the decomposition of generalized additive independence models. page 26.

[Grabisch and Labreuche, 2018] Grabisch, M. and Labreuche, C. (2018). Monotone decomposition of 2-additive Generalized Additive Independence models. *Mathematical Social Sciences*, 92:64–73.

[Grabisch et al., 2009] Grabisch, M., Marichal, J.-L., Mesiar, R., and Pap, E. (2009). *Aggregation Functions*. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

[Grabisch et al., 2000] Grabisch, M., Murofushi, T., Sugeno, M., and Kacprzyk, J. (2000). *Fuzzy Measures and Integrals. Theory and Applications*. Physica Verlag, Berlin.

[Grabisch and Perny, 2003] Grabisch, M. and Perny, P. (2003). Agrégation multicritère.

[Greco et al., 2005] Greco, S., Mousseau, V., and Slowinski, R. (2005). Ordinal regression revisited: multiple criteria ranking with a set of additive value functions. page 30.

[Greco et al., 2009] Greco, S., Slowinski, R., Figueira, J. R., and Mousseau, V. (2009). Robust Ordinal Regression. page 54.

[Havens and Anderson, 2018] Havens, T. C. and Anderson, D. T. (2018). Machine Learning of Choquet Integral Regression with Respect to a Bounded Capacity (or Non-monotonic Fuzzy Measure). In *FUZZ-IEEE-18*.

[Hein and Andriushchenko, 2017] Hein, M. and Andriushchenko, M. (2017). Formal guarantees on the robustness of a classifier against adversarial manipulation. *arXiv:1705.08475 [cs, stat]*.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

[Huang et al., 2016] Huang, X., Kwiatkowska, M., Wang, S., and Wu, M. (2016). Safety verification of deep neural networks. *CoRR*, abs/1610.06940.

[Huang et al., 2008] Huang, Z., Gedeon, T., and Nikravesh, M. (2008). Pattern tree induction: A new machine learning method. *IEEE TFS*, 16(4):958–970.

[Hüllermeier and Fallah Tehrani, 2012] Hüllermeier, E. and Fallah Tehrani, A. (2012). Efficient learning of classifiers based on the 2-additive Choquet integral. In Moewes, C. and Nürnberger, A., editors, *Computational Intelligence in Intelligent Data Analysis*, Studies in Computational Intelligence, pages 17–30. Springer.

[Hüllermeier and Fallah Tehrani, 2012] Hüllermeier, E. and Fallah Tehrani, A. (2012). On the vc-dimension of the choquet integral. volume 297, pages 42–50.

[Hüllermeier et al., 2008] Hüllermeier, E., Fürnkranz, J., Cheng, W., and Brinker, K. (2008). Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16):1897–1916.

[Hüllermeier and Tehrani, 2013] Hüllermeier, E. and Tehrani, A. F. (2013). Efficient Learning of Classifiers Based on the 2-Additive Choquet Integral. In Moewes, C. and Nürnberger, A., editors, *Computational Intelligence in Intelligent Data Analysis*, Studies in Computational Intelligence, pages 17–29. Springer Berlin Heidelberg.

[Ilyas et al., 2019] Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. (2019). Adversarial examples are not bugs, they are features.

[Jiang et al., 2018] Jiang, H., Kim, B., Guan, M. Y., and Gupta, M. (2018). To trust or not to trust a classifier. *arXiv:1805.11783 [cs, stat]*.

[Katz et al., 2017] Katz, G., Barrett, C. W., Dill, D. L., Julian, K., and Kochenderfer, M. J. (2017). Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, abs/1702.01135.

[Keeney et al., 1979] Keeney, R., Raiffa, H., and W. Rajala, D. (1979). Decisions with multiple objectives: Preferences and value trade-offs. *Systems, Man and Cybernetics, IEEE Transactions on*, 9:403–403.

[Khan et al., 2019] Khan, A., Sohail, A., Zahoora, U., and Qureshi, A. S. (2019). A survey of the recent architectures of deep convolutional neural networks. *CoRR*, abs/1901.06032.

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

[Krantz et al., 1971] Krantz, D., Luce, R., Suppes, P., and Tversky, A. (1971). *Foundations of measurement*, volume 1: Additive and Polynomial Representations. Academic Press.

[Labreuche, 2018] Labreuche, C. (2018). An axiomatization of the Choquet integral and its utility functions without any commensurability assumption. *Annals of Operation Research*, 271(2):701–735.

[Labreuche and Fossier, 2018] Labreuche, C. and Fossier, S. (2018). Explaining multi-criteria decision aiding models with an extended shapley value. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 331–339. International Joint Conferences on Artificial Intelligence Organization.

[Labreuche and Grabisch, 2013] Labreuche, C. and Grabisch, M. (2013). Use of the GAI model in multi-criteria decision making: inconsistency handling, interpretation.

[Labreuche et al., 2016] Labreuche, C., Hüllermeier, E., Vojtas, P., and Tehrani, A. F. (2016). On the Identifiability of Models in Multi-Criteria Preference Learning. In *DA2PL-16*.

[Le Cun et al., 1989] Le Cun, Y., Denker, J. S., and Solla, S. A. (1989). Optimal brain damage. *Advances in Neural Information Processing Systems 2 (NIPS 1989)*.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Leroy et al., 2011] Leroy, A., Mousseau, V., and Pirlot, M. (2011). Learning the parameters of a multiple criteria sorting method. In Brafman, R. I., Roberts, F. S., and Tsoukiàs, A., editors, *Algorithmic Decision Theory*, volume 6992, pages 219–233. Springer Berlin Heidelberg.

[Liu, 2009] Liu, T.-Y. (2009). Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331.

[Liu et al., 2017] Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., and Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26.

[Machida, 2019] Machida, F. (2019). N-version machine learning models for safety critical systems. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 48–51. IEEE.

[Madry et al., 2018] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. *CoRR*, abs/1706.06083.

[Martin and Perny, 2020] Martin, H. and Perny, P. (2020). New Computational Models for the Choquet Integral. In *24th European Conference on Artificial Intelligence - ECAI 2020*, Santiago, Spain.

[Mayag et al., 2011] Mayag, B., Grabisch, M., and Labreuche, C. (2011). A representation of preferences by the choquet integral with respect to a 2-additive capacity. *Theory and Decision*, 71(3):297–324.

[Miller, 1956] Miller, G. A. (1956). The magical number seven plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63 2:81–97.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.

[Modave and Grabisch, 1998] Modave, F. and Grabisch, M. (1998). Preference representation by the choquet integral: The commensurability hypothesis. *IPMU 1998*.

[Murofushi and Soneda, 1993] Murofushi, T. and Soneda, S. (1993). Techniques for reading fuzzy measures (III): interaction index. In *9th Fuzzy System Symposium*, pages 693–696, Sapporo, Japan.

[Murofushi and Sugeno, 1989] Murofushi, T. and Sugeno, M. (1989). An interpretation of fuzzy measures and the choquet integral as an integral with respect to a fuzzy measure. *Fuzzy Sets and Systems*, 29:201–227.

[Nguyen et al., 2019] Nguyen, V.-L., Destercke, S., and Hüllermeier, E. (2019). Epistemic uncertainty sampling. *arXiv:1909.00218 [cs, stat]*.

[Nguyen et al., 2018] Nguyen, V.-L., Destercke, S., Masson, M.-H., and Hüllermeier, E. (2018). Reliable multi-class classification based on pairwise epistemic and aleatoric uncertainty. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 5089–5095. International Joint Conferences on Artificial Intelligence Organization.

[Nwankpa et al., 2018] Nwankpa, C., Ijomah, W. L., Gachagan, A., and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *ArXiv*, abs/1811.03378.

[O'Neill, 2016] O'Neill, C. (2016). *Weapons of Math Destruction*. Crown Books.

[Ovchinnikov, 2002] Ovchinnikov, S. (2002). Max-min representation of piecewise linear functions. *Beiträge zur Algebra und Geometrie*, 43.

[Pastijn and Leysen, 1989] Pastijn, H. and Leysen, J. (1989). Constructing an outranking relation with oreste. page 14.

[Paulino and Pereira, 1994] Paulino, C. and Pereira, C. (1994). On identifiability of parametric statistical models. *Journal of the Italian Statistical Society*, 3:125–151.

[Pelissari and Duarte, 2020] Pelissari, R. and Duarte, L. T. (2020). Identification of choquet capacity in multicriteria sorting problems through stochastic inverse analysis. *arXiv:2003.12530 [cs, stat]*. arXiv: 2003.12530.

[Perny et al., 2016] Perny, P., Viappiani, P., and Boukhatem, A. (2016). Incremental preference elicitation for decision making under risk with the rank-dependent utility model. In *UAI*.

[Pulina and Tacchella, 2010] Pulina, L. and Tacchella, A. (2010). An abstraction-refinement approach to verification of artificial neural networks. In *CEUR Workshop Proceedings*, volume 616, pages 243–257.

[Qian et al., 2015] Qian, L., Gao, J., and Jagadish, H. V. (2015). Learning user preferences by adaptive pairwise comparison. *Proc. VLDB Endow.*, 8(11):1322–1333.

[Ran and Hu, 2017] Ran, Z.-Y. and Hu, B.-G. (2017). Parameter identifiability in statistical machine learning: A review. *Neural Computation*, 29(5):1151–1203.

[Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386.

[Rota, 1964] Rota, G.-C. (1964). On the foundations of combinatorial theory i. theory of moebius functions. page 29.

[Rothenberg, 1971] Rothenberg, T. J. (1971). Identification in parametric models. *Econometrica*, 39(3):pp. 577–591.

[Roubens, 1982] Roubens, M. (1982). Preference relations on actions and criteria in multicriteria decision making. *European Journal of Operational Research*, 10(1):51–55.

[Roy, 1996] Roy, B. (1996). *Multicriteria Methodology for Decision Aiding*. Nonconvex Optimization and Its Applications. Springer US.

[Roy, 1999] Roy, B. (1999). Decision aiding today: what should we expect? page 35.

[Ruder, 2017] Ruder, S. (2017). An overview of gradient descent optimization algorithms.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

[Saint-Hilary et al., 2017] Saint-Hilary, G., Cadour, S., Robert, V., and Gasparini, M. (2017). A simple way to unify multicriteria decision analysis (MCDA) and stochastic multicriteria acceptability analysis (SMAA) using a dirichlet distribution in benefit–risk assessment. *Biometrical Journal*, 59(3):567–578.

[Senge et al., 2014] Senge, R., Bösner, S., Dembczyński, K., Haasenritter, J., Hirsch, O., Donner-Banzhoff, N., and Hüllermeier, E. (2014). Reliable classification: Learning classifiers that distinguish aleatoric and epistemic uncertainty. *Information Sciences*, 255:16–29.

[Senge and Hüllermeier, 2011] Senge, R. and Hüllermeier, E. (2011). Top-Down Induction of Fuzzy Pattern Trees. *IEEE Transactions on Fuzzy Systems*, 19(2):241–252.

[Shafer, 1976] Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press, Princeton.

[Shapley, 1953] Shapley, L. S. (1953). A value for n-person games. In Kuhn, H. W. and Tucker, A. W., editors, *Contributions to the Theory of Games, Vol. II*, number 28 in Annals of Mathematics Studies, pages 307–317. Princeton University Press.

[Singh et al., 2019] Singh, G., Gehr, T., Püschel, M., and Vechev, M. (2019). An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3:1–30.

[Siskos, 1982] Siskos, J. (1982). Assessing a set of additive utility functions for multicriteria decision-making, the UTA method.

[Siskos et al., 2005] Siskos, Y., Grigoroudis, E., Matsatsinis, N., Figueira, J., Greco, S., and Ehrogott, M. (2005). *UTA methods*, volume 233, pages 297–334.

[Sobrie et al., 2015] Sobrie, O., Mousseau, V., and Pirlot, M. (2015). Learning the parameters of a non compensatory sorting model. pages 153–170.

[Sobrie et al., 2016] Sobrie, O., Mousseau, V., and Pirlot, M. (2016). Learning MR-sort rules with coalitional veto. page 9.

[Song and Kingma, 2021] Song, Y. and Kingma, D. P. (2021). How to train your energy-based models. *CoRR*, abs/2101.03288.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

[Stewart et al., 2005] Stewart, T., Figueira, J., Greco, S., and Ehrogott, M. (2005). Dealing with uncertainties in MCDA. pages 445–466.

[Sugeno, 1974] Sugeno, M. (1974). *Theory of fuzzy integrals and its applications*. phdthesis.

[Szegedy et al., 2013] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv:1312.6199 [cs]*.

[Tehrani et al., 2014] Tehrani, A. F., Labreuche, C., and Hüllermeier, E. (2014). Choquistic utilitaristic regression. In *Decision Aid to Preference Learning (DA2PL) workshop*, Chatenay-Malabry, France.

[Valiant, 1984] Valiant, L. G. (1984). A theory of the learnable. *Commun. ACM*, 27(11):1134–1142.

[Vapnik, 1998] Vapnik, V. N. (1998). *Statistical Learning Theory.* Wiley-Interscience.

[Varshney and Alemzadeh, 2016] Varshney, K. R. and Alemzadeh, H. (2016). On the safety of machine learning: Cyber-physical systems, decision sciences, and data products. *arXiv:1610.01256 [cs, stat].*

[Wachsmuth et al., 2017] Wachsmuth, H., Naderi, N., Hou, Y., Bilu, Y., Prabhakaran, V., Thijm, T., Hirst, G., and Stein, B. (2017). Computational argumentation quality assessment in natural language. In *Proc. 15th Conf. of the European Chapter of the Ass. for Computational Linguistics.*

[Wang et al., 2018] Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. (2018). Efficient formal safety analysis of neural networks. *CoRR*, abs/1809.08098.

[Winter, 1989] Winter, E. (1989). A value for cooperative games with levels structure of cooperation. *International Journal of Game Theory*, 18(2):227–40.

[Wolpert and Macready, 1997] Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

[Yager, 1988] Yager, R. R. (1988). On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):183–190.

[Yang et al., 2021] Yang, Z., Zhang, A., and Sudjianto, A. (2021). Gami-net: An explainable neural network based on generalized additive models with structured interactions.

[Zeiler, 2012] Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.

[Zhou, 2012] Zhou, Z.-H. (2012). *Ensemble Methods: Foundations and Algorithms.* Chapman & Hall/CRC, 1st edition.