Learning capacities from data in large universes

by

Vincent Racine

A research project
presented to the University of Waterloo
in fulfillment of the
research paper requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Waterloo, Ontario, Canada, 2023

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The Choquet integral, defined with respect to a capacity, also known as a non-additive set function or fuzzy measure, constitutes a versatile class of aggregation operators. Unfortunately, the application of the Choquet integral has been predominantly constrained to problems with a relatively small number of inputs. This limitation arises from the fact that for n inputs, the corresponding fuzzy measure involves 2^n variables subject to $n(2^n - 1)$ monotonicity constraints.

In this study, we introduce an innovative algorithm based on neural networks to estimate a fuzzy measure from data by learning its associated pseudo-Boolean function. This approach opens a realm of possibilities as it exhibits the potential to overcome the two main challenges encountered when attempting to directly learn the coefficients of the capacity. Firstly, it allows for the selection of a parameter count p in the neural network that is significantly lower than the 2^n coefficients required for complete capacity identification, while still retaining robust modeling capabilities. Secondly, enforcing the monotonicity constraint is simplified by ensuring non-negative weights within the neural network. As a result, the proposed method demonstrates scalability and the ability to represent and learn the Choquet integral in dimensions of up to 100 variables. This represents a substantial advancement over the current state-of-the-art techniques. Numerical experiments conducted on synthetic datasets with known capacities further validate the effectiveness of our proposed methodology.

Acknowledgements

I extend my sincere gratitude to my supervisors, Prof. Mario Ghossoub and Prof. David Saunders, for their unwavering guidance and invaluable support throughout the entirety of this project. Additionally, I wish to express my appreciation to Prof. Bin Li, the second reader, for his constructive feedback.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The utilization of fuzzy integrals for input aggregation opens up numerous possibilities for modeling interaction, redundancy, and complementarity among inputs. Unlike conventional aggregation strategies such as weighted, power, or geometric means and medians, which treat inputs—such as the degree of satisfaction of criteria or features in machine learning—independently, fuzzy integrals assign weights to input subsets to generate representative outputs. This approach allows inputs to gain importance when combined as a coalition and avoid double counting redundant inputs. The Choquet integral [17], a prominent fuzzy integral, has found extensive application in pattern recognition [70], computer vision [55, 53, 59], classification [63, 26, 30], multi-criteria decision making [33], model ensemble [69, 38, 66, 54, 12, 24], and more.

Fuzzy integrals are defined in relation to a fuzzy measure, also referred to as a capacity. Capacities are monotone, grounded, and normalized set functions [17, 31] that assign numerical weights to input coalitions, reflecting their relative importance. The flexibility of fuzzy measures in modeling input interactions, however, comes with a notable trade-off—complexity. For n inputs, a fuzzy measure entails 2^n coefficients, subject to $n(2^n - 1)$ monotonicity constraints. Consequently, empirical data becomes essential for constructing fuzzy measures. The learning problem involves fitting a Choquet integral-based model to input-output data pairs. Various algorithms have emerged for this purpose, including linear programming (LP) [6], quadratic programming (QP) [40, 34], gradient descent [29, 65], genetic algorithms [68, 35], particle swarm optimization [69], and Gibbs samplers [47].

Nonetheless, the complexity of fuzzy measures poses a substantial challenge to the learning process. The sheer number of variables requiring identification and the monotonicity constraints render the learning problem computationally demanding for modest $n \geq 10$.

Thus, simplification is necessary to alleviate the complexity of fuzzy measures. Drastic simplifications, such as imposing symmetry and additivity, lead to well-known ordered weighted average (OWA) and weighted average mean (WAM) approaches. However, these methods do not account for interactions between inputs. λ-fuzzy measures [62], requiring specification of only $n + 1$ parameters (densities and the λ parameter), are confined to totally-alternating or totally monotone capacities, limiting their modeling power. Another approach involves constraining the capacity to a 2-additive measure [30], restricts interactions to pairs of inputs. While this simplification reduces problem complexity, it does not encompass higher-order interactions crucial in domains like machine learning, signal/image processing, and computer vision.

In this research project, we introduce an innovative approach—employing deep neural networks to learn the pseudo-Boolean function associated with the optimal capacity. This approach offers dual advantages: firstly, we can choose the number of parameters p in the neural network to be much smaller than the capacity's 2^n, while maintaining robust modeling capabilities; secondly, enforcing non-negative weights in the neural network ensures that the associated capacity is monotone. By leveraging these advantages, we efficiently learn capacities involving up to 100 variables, marking a substantial leap beyond current state-of-the-art methods.

It's important to acknowledge that we are not the pioneers in proposing the utilization of deep neural networks to learn capacities from data [38, 67]. The primary innovation of our approach lies in recommending the learning of the pseudo-Boolean function linked to the optimal capacity, in contrast to prior works that employed a neural network to directly learn the coefficients of the capacity. Under their formulation, they are compelled to establish the count of learnable parameters p within the neural network equivalent to the number required for identifying a capacity, i.e., 2^n. Consequently, their suggested methodology faces limitations in scalability as the size of the universe increases.

Many modern practical applications involve large universal sets. For instance, it is not uncommon to encounter universes with hundreds of inputs when performing feature and signal fusion in machine learning [55]. Numerous other applications with extensive universes are highlighted in [7]. We believe that the proposed method offers an efficient means to represent and identify fuzzy measures in such scenarios. Although not explored in this project, our approach presents additional benefits, including enhanced flexibility in loss function selection [65], and the potential for concurrent incorporation and training as a layer within a deep neural network architecture [38].

The structure of the remaining chapters in this research paper is as follows. Chapter 2 delves into essential background information, focusing on key aspects such as fuzzy mea-

sures, the Choquet integral, pseudo-Boolean functions, and, notably, the intricate challenge of learning capacities from data. In Chapter 3, we redefine the task of capacity identification from data as the identification of the corresponding pseudo-Boolean function. We propose a novel approach that employs deep feed-forward neural networks for modeling the pseudo-Boolean function, and we describe methods to ensure the associated capacity is monotone, grounded, and normalized. The outcomes of our numerical experiments, evaluating the efficiency and accuracy of our proposed method, are presented in Chapter 4. Finally, Chapter 5 provides concluding remarks.

Chapter 2

Background

This chapter establishes the foundational context for our research project. In Section 2.1, we lay the essential groundwork concerning fuzzy measures. Section 2.2 delves into the fundamental concept of the Choquet Integral. In Section 2.3, we explore pseudo-Boolean functions and their intricate relationship with set functions. Section 2.4 formalizes the challenge of learning capacities from empirical data. Additionally, in Section 2.5, we review relevant literature that aligns with the scope of this research endeavor.

It is important to note that the content presented in sections 2.1, 2.2, and 2.3 is of a relatively elementary nature and is readily accessible at the textbook level, exemplified by references such as [31, 9].

In the context of this research project, our focus is solely on discrete fuzzy measures, which find application within finite discrete subsets. This concentration stems from the fact that aggregation functions involve a finite set of inputs. While the Choquet integral can be defined for measures encompassing general sets, a comprehensive review of the extensive literature pertaining to this subject falls beyond the purview of this project. Furthermore, it is pertinent to acknowledge that the definitions provided for fuzzy measures in Section 2.1 hold a broader applicability to games (grounded set function).

Throughout the ensuing sections, we shall adhere to the following notation: μ denotes a capacity, ξ refers to a general set function, and boldface $\mathbf{x}$ signifies that x is a vector.

For ease of reference and coherence, we establish the finite discrete set $\mathcal{N} = \{1, ..., n\}$ as a constant framework throughout the remainder of this paper.

2.1 Fuzzy Measure

A fuzzy measure, also known as a capacity, is a monotone, grounded, and normalized set function. We denote the collection of all normalized capacities defined on the power set $2^{\mathcal{N}}$ as $\mathcal{MG}_0(\mathcal{N})$.

Definition 2.1.1. *(Discrete fuzzy measure) A fuzzy measure $\mu : 2^{\mathcal{N}} \mapsto [0,1]$ is a set function that satisfies the following three properties:*

1. *(Monotone) $\mu(A) \leq \mu(B)$ whenever $A \subseteq B$,*
2. *(Grounded) $\mu(\emptyset) = 0$,*
3. *(Normalized) $\mu(\mathcal{N}) = 1$.*

Remark 2.1.1. *(Normalization) Within certain literature, a capacity is defined as a set function that exhibits monotonicity and groundedness. When a set function possesses normalization alongside monotonicity and groundedness, it is termed a normalized (or regular) capacity. Nevertheless, in the context of this research project, we will strictly follow definition 2.1.1. Thus, we will use the term "capacity" to specifically signify a set function that fulfills the criteria of being monotone, grounded, and normalized.*

Additionally, the dual or conjugate fuzzy measure $\bar{\mu}$ is defined in Definition 2.1.2.

Definition 2.1.2. *(Dual Fuzzy Measure) Given a fuzzy measure μ, its dual (also known as its conjugate) $\bar{\mu}$ is defined by*

$$\bar{\mu}(A) = \mu(\mathcal{N}) - \mu(A^c), \tag{2.1}$$

where $A^c = \mathcal{N} \setminus A$ is the complement of A in $\mathcal{N}$.

The Möbius transform, outlined in Definition 2.1.3, serves as a linear transformation of μ.

Definition 2.1.3. *(Möbius Transform) The Möbius transform of μ, denoted as m^μ, is a set function defined by*

$$m^\mu(A) = \sum_{B \subseteq A} (-1)^{|A \setminus B|} \mu(B) \ \forall A \subseteq \mathcal{N}. \tag{2.2}$$

The Möbius transform can be inverted using the Zeta function, as indicated by Equation (2.3):

$$\mu(A) = \sum_{B \subseteq A} m^{\mu}(B) \ \forall A \subseteq \mathcal{N}. \tag{2.3}$$

The concept of derivative for a fuzzy measure is defined in Definition 2.1.4, while second-order derivatives are introduced in Definition 2.1.5.

Definition 2.1.4. *(Derivative of a fuzzy measure) Let μ be a capacity. The derivative of μ at $A \subseteq \mathcal{N}$ with respect to $i \in \mathcal{N}$ is defined as*

$$\Delta_i \mu(A) = \mu(A \cup \{i\}) - \mu(A). \tag{2.4}$$

Remark 2.1.2. *For monotone set functions the derivative is always non-negative $\Delta_i \mu(A) \geq 0$.*

Definition 2.1.5. *(Second order Derivative of a fuzzy measure) The second order derivative of μ at $A \subseteq \mathcal{N}$ with respect to $i, j \in \mathcal{N}$ is defined as*

$$\Delta_{ij} \mu(A) = \mu(A \cup \{i, j\}) - \mu(A \cup \{i\}) - \mu(A \cup \{j\}) + \mu(A). \tag{2.5}$$

Notably, fuzzy measures can possess properties such as superadditivity and subadditivity (Definition 2.1.6), supermodularity and submodularity (Definition 2.1.7), as well as k-monotonicity and k-alternation (Definition 2.1.8). Furthermore, maxitive and minitive properties are defined in Definition 2.1.9.

Definition 2.1.6. *(Superadditive and Subadditive fuzzy measure) Let μ be a fuzzy measure. We say that μ is supperadditive if for all disjoint $A, B \subseteq \mathcal{N}$*

$$\mu(A \cup B) \geq \mu(A) + \mu(B).$$

It is subadditive if the reverse inequality holds

$$\mu(A \cup B) \leq \mu(A) + \mu(B).$$

Definition 2.1.7. *(Supermodular and Submodular fuzzy measure) A fuzzy measure μ is supermodular if for all $A, B \subseteq \mathcal{N}$*

$$\mu(A \cup B) \geq \mu(A) + \mu(B) - \mu(A \cap B).$$

It is submodular if the reverse inequality holds

$$\mu(A \cup B) \leq \mu(A) + \mu(B) - \mu(A \cap B).$$

Remark 2.1.3. *Supermodularity (Submodularity) implies superadditivity (subadditivity), but the converse is not true.*

Definition 2.1.8. *(k-monotone and k-alternating fuzzy measure) A fuzzy measure μ is k-monotone ($k \geq 2$) if for any family of k sets $A_1, ..., A_k \in 2^{\mathcal{N}}$*

$$\mu(\cup_{i=1}^{k} A_i) \geq \sum_{\substack{I \subseteq \{1,...,k\} \\ I \neq \emptyset}} (-1)^{|I|+1} \mu(\cap_{i \in I} A_i).$$

A fuzzy measure is totally-monotone if it is k-monotone for any $k \geq 2$. A fuzzy measure is k-alternating if

$$\mu(\cap_{i=1}^{k} A_i) \leq \sum_{\substack{I \subseteq \{1,...,k\} \\ I \neq \emptyset}} (-1)^{|I|+1} \mu(\cup_{i \in I} A_i),$$

and totally-alternating if it is k-alternating for any $k \geq 2$.

Remark 2.1.4. *2-alternating is equivalent to submodular and 2-monotone is equivalent to supermodular.*

Definition 2.1.9. *(Maxitive and minitive) Let μ be a normalized capacity. We say that μ is maxitive if for all $A, B \subseteq \mathcal{N}$*

$$\mu(A \cup B) = \max(\mu(A), \mu(B)),$$

and that μ is minitive if

$$\mu(A \cap B) = \min(\mu(A), \mu(B)).$$

To define a fuzzy measure, one typically needs to specify values for all possible coalitions of inputs, resulting in a large number of parameters $(2^n - 2)$ that needs to be identified. However, there are certain families of fuzzy measures that exhibit interdependencies among these coefficients, leading to a reduction in the number of independent parameters. The following definitions highlight some such families:

Definition 2.1.10. *(Boolean measure) A fuzzy measure μ is called a boolean fuzzy measure or a 0-1 fuzzy measure if it holds that*

$$\mu(A) = 0 \text{ or } \mu(A) = 1 \ \forall A \subseteq \mathcal{N}.$$

Definition 2.1.11. *(Additive) Let μ be a fuzzy measure. We say that μ is additive if for all disjoint sets $A, B \subseteq \mathcal{N}$*

$$\mu(A \cup B) = \mu(A) + \mu(B).$$

An additive capacity is a probability measure.

Definition 2.1.12. *(Symmetric) A fuzzy measure μ is said to be symmetric if its value depends only on the cardinality of the set, i.e. $\mu(A) = \mu(B)$ whenever $|A| = |B|$.*

Definition 2.1.13. *(λ-fuzzy measure) Given a parameter $\lambda \in (-1, \infty)$, a λ-fuzzy measure is a fuzzy measure μ, which for all disjoint sets $A, B \in \mathcal{N}$*

$$\mu(A \cup B) = \mu(A) + \mu(B) + \lambda\mu(A)\mu(B). \tag{2.6}$$

When μ is a λ-fuzzy measure, the value of $\mu(A)$ for all $A \subseteq \mathcal{N}$ can be explicitly computed from the value on its singletons $\mu(\{i\})$ for $i \in \mathcal{N}$

$$\mu(A) = \frac{1}{\lambda}\left(\prod_{i \in A}(1 + \lambda\mu(\{i\})) - 1\right) \quad \lambda \neq 0. \tag{2.7}$$

If $\lambda = 0$ then μ is a probability measure. The value for the coefficient λ is determined from the boundary condition $\mu(\mathcal{N}) = 1$, which gives

$$1 + \lambda = \prod_{i \in \mathcal{N}}(1 + \lambda\mu(\{i\})). \tag{2.8}$$

Equation 2.8 can be solved numerically on the interval $(-1, 0)$ or $(0, \infty)$ (λ=0 is always a solution). When $-1 \leq \lambda \leq 0$ the λ-fuzzy measure is totally-alternating and when $\lambda \geq 0$ the λ-fuzzy measure is totally-monotone.

2.2 Choquet Integral

The Choquet integral with respect to a capacity μ is defined as follows:

Definition 2.2.1. *(Discrete Choquet integral) The discrete Choquet integral with respect to a fuzzy measure μ is given by*

$$C_\mu(\mathbf{x}) = \sum_{i=1}^{n} x_{\sigma_i}(\mu(H_i) - \mu(H_{i+1})), \tag{2.9}$$

where $\mathbf{x}_\nearrow = (x_{\sigma_1}, x_{\sigma_2}, ..., x_{\sigma_n})$ is a non-decreasing permutation of the input $\mathbf{x}$ and $H_i = \{\sigma_i, ..., \sigma_n\}$ and $H_{n+1} = \emptyset$.

A geometric interpretation of the Choquet integral can be elucidated. We can divide the set $[0,1]^n$ into $n!$ polytopes of the form $\{\mathbf{x} \in [0,1]^n : x_{\sigma_1} \leq ... \leq x_{\sigma_n}\}$, with one polytope corresponding to one permutation σ. For a given input $\mathbf{x} \in [0,1]^n$, the Choquet integral becomes a linear interpolation of the vertices of its corresponding polytope. A visual representation can be found in Figure 3.2 and 3.3 of [5].

The Choquet integral can also be expressed in terms of its Mobius transform:

$$C_\mu(\mathbf{x}) = \sum_{A \subseteq \mathcal{N}} m^\mu(A) \min_{i \in A} x_i. \tag{2.10}$$

Utilizing Definition 2.1.3, we can demonstrate that:

$$C_\mu(\mathbf{x}) = \sum_{A \subseteq \mathcal{N}} \mu(A) K_{\mathbf{x}}(A), \tag{2.11}$$

where $K_{\mathbf{x}}(A)$ is defined as:

$$K_{\mathbf{x}}(A) = \sum_{B \supseteq A} (-1)^{|A \setminus B|} \min_{i \in B} x_i = \max(0, \min_{i \in A} x_i - \min_{i \in A^c} x_i), \tag{2.12}$$

with the convention that $\max_{i \in \emptyset} x_i = \min_{i \in \emptyset} x_i = 0$.

The subsequent definitions prove to be valuable while establishing the properties of the Choquet integral:

Definition 2.2.2. *(Comonotone) We say that two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ are comonotone if there exists a common permutation σ such that $x_{\sigma_1} \leq ... \leq x_{\sigma_n}$ and $y_{\sigma_1} \leq ... \leq y_{\sigma_n}$. Equivalently, this condition is often expressed as $(x_i - x_j)(y_i - y_j) \geq 0$ for all $i, j \in \mathcal{N}$.*

Definition 2.2.3. *(Dominance relationship) Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. We say that $\mathbf{x}$ is dominated by $\mathbf{y}$, denoted as $\mathbf{x} \preccurlyeq \mathbf{y}$, when $x_i \leq y_i$ for all $i \in \mathcal{N}$.*

The Choquet Integral exhibits the following notable properties:

- The Choquet integral is comonotone additive meaning that if $\mathbf{x}$ and $\mathbf{y}$ are comonotone then $C_\mu(\mathbf{x} + \mathbf{y}) = C_\mu(\mathbf{x}) + C_\mu(\mathbf{y})$.
- The Choquet integral is a monotone function; if $\mathbf{x} \preccurlyeq \mathbf{y}$ then $C_\mu(\mathbf{x}) \leq C_\mu(\mathbf{y})$.

- The Choquet integral is positively homogeneous, i.e., for all $c \geq 0$ and $\mathbf{x} \in \mathbb{R}^n$ $C_\mu(c\mathbf{x}) = cC_\mu(\mathbf{x})$.
- The Choquet integral is a convex function if and only if the underlying fuzzy measure is submodular.

Remark 2.2.1. *(The many fuzzy integrals) There exist a number of fuzzy integrals defined on discrete subsets. In this research project, our focus is on the Choquet integral, as it is the most commonly used fuzzy integral. However, we want to emphasize that our proposed method could be adapted to other fuzzy integrals, as long as one can compute a (sub)-derivative of the fuzzy integral with respect to the parameters of the capacity μ. For a deeper exploration of fuzzy integrals, we recommend consulting [9].*

2.3 Pseudo-Boolean Function

Pseudo-boolean functions are functions defined on the vertices of a hypercube. There exists a one-to-one mapping between set-functions defined on $2^{\mathcal{N}}$ and pseudo-boolean functions defined on $\{0,1\}^n$. This relationship is established as follows:

Given a set function ξ, a unique pseudo-boolean function f_ξ is defined by:

$$f_\xi(\mathbf{x}) = \xi(i \mid \mathbf{x_i} = 1) \text{ for all } \mathbf{x} \in \{0,1\}^n; \tag{2.13}$$

Conversely, for any pseudo-boolean function, a unique set function ξ can be defined by:

$$\xi(A) = f_\xi(\mathbf{1_A}) \text{ for all } A \subseteq \mathcal{N}. \tag{2.14}$$

This correspondence allows for the transposition of any definitions or properties established on set functions to pseudo-boolean functions, and vice versa. For instance, pseudo-Boolean functions, which are monotone, grounded ($f_\xi(\mathbf{0}) = 0$), and normalized ($f_\xi(\mathbf{1}) = 1$) set functions, are associated with fuzzy measures.

A pseudo-boolean function is considered monotone if $f(\mathbf{1}_A) \leq f(\mathbf{1}_B)$ whenever the characteristic vector $\mathbf{1}_A$ is dominated by $\mathbf{1}_B$.

We will use the notation $\mathcal{PB}(n)$ to represent the set of all pseudo-boolean functions defined on $\{0,1\}^n$. This set forms a vector space with bases. One common choice for the basis is the set of polynomials:

$$\prod_{i \in A} x_i \prod_{i \in A^c} (1 - x_i) \text{ for all } A \subseteq \mathcal{N}, \tag{2.15}$$

where $\prod_{i\in\emptyset} x_i = 1$ is conventionally defined. With this basis, a pseudo-boolean function f can be expressed as a polynomial:

$$f(\mathbf{x}) = \sum_{A\subseteq\mathcal{N}} \theta_A \prod_{i\in A} x_i \prod_{i\in A^c} (1 - x_i). \tag{2.16}$$

Here, the coefficients $\theta \in \mathbb{R}^{2^n}$ are treated as parameters. This representation is referred to as the standard representation of f. Given that $\prod_{i\in A} x_i \prod_{i\in A^c}(1 - x_i)$ are Dirac set functions, such that:

$$\prod_{i\in A} x_i \prod_{i\in A^c} (1 - x_i) = \begin{cases} 1 \text{ if } \mathbf{x} = \mathbf{1}_A, \\ 0 \text{ otherwise,} \end{cases} \tag{2.17}$$

it follows from equation (2.16) that $f(\mathbf{1_A}) = \theta_A = \mu(A)$ for all $A \subseteq \mathcal{N}$.

An alternative basis consists of monomials corresponding to unanimity games:

$$\prod_{i\in A} x_i \text{ for all } A \in \mathcal{N}. \tag{2.18}$$

This leads to another polynomial representation of pseudo-boolean functions:

$$f(\mathbf{x}) = \sum_{A\subseteq\mathcal{N}} \theta_A \prod_{i\in A} x_i. \tag{2.19}$$

Under this representation, the coefficients θ_A correspond to the Möbius transform of the set function ξ associated with the pseudo-boolean function f. The Möbius representation can be especially advantageous when the objective is to approximate polynomials with a degree no greater than $k < n$.

2.4 Problem Statement

In this research project, our focus lies in fitting Choquet-integral based models to numerical data. The fundamental problem at hand is to determine the underlying capacity μ that best captures the relationship between a dataset of m input-output pairs. These pairs consist of observations (or feastures) $X \in [0, 1]^{m\times n}$ and corresponding targets $\mathbf{y} \in [0, 1]^m$. The objective is to find a capacity μ whose Choquet integral $C_\mu(X) \in [0, 1]^m$ yields the most accurate predictions for the target values $\mathbf{y}$.

We can frame the task of fitting Choquet-integral based models to data as an optimization problem, denoted as follows:

$$\begin{aligned} \min_{\mu} \quad & \mathcal{L}(\mu) \\ \text{s.t.} \quad & \mu(A) \leq \mu(A \cup \{i\}) \text{ for all } A \subseteq \mathcal{N}, i \notin A, \\ & \mu(\emptyset) = 0, \\ & \mu(\mathcal{N}) = 1. \end{aligned} \tag{2.20}$$

Here, $\mathcal{L}(\mu)$ represents the loss function, which quantifies the discrepancy between the predicted Choquet integral $C_\mu(X)$ and the target $\mathbf{y}$. For instance, in a least squared fitting scenario, the loss function corresponds to the l_2 norm

$$\mathcal{L}(\mu) = ||C_\mu(X) - \mathbf{y}||_2^2. \tag{2.21}$$

In this optimization problem, the variables to be determined are the coefficients of the capacity $\mu(A)$ for all $A \subseteq \mathcal{N}$. The constraints ensure that μ belongs to the compact, convex, polyhedron $\mathcal{MG}_0(\mathcal{N})$ of capacities defined on the power set $2^{\mathcal{N}}$.

Since the Choquet integral exhibits linearity with respect to μ (using the basis functions $K_{\mathbf{x}}(A)$ defined in equation (2.12)), problem (2.20) becomes a Quadratic Program (QP) when adopting a least squared fitting approach. Alternatively, it transforms into a Linear Program (LP) when employing a least absolute deviation fitting criterion. Consequently, standard techniques for solving QPs and LPs can be applied to address problem (2.20) for small input sizes n, as demonstrated in works such as [34, 8].

However, as the number of variables and constraints grows exponentially with the input size n, addressing problem (2.20) for larger input sizes requires the utilization of heuristics and simplifying assumptions. In the subsequent section, we will delve into a comprehensive review of these strategies, which play a crucial role in enabling the effective solution of the problem in scenarios involving larger input dimensions.

2.5 Related Work

One of the most well-established algorithms for learning fuzzy measures from empirical data is the Heuristic Least Mean Square (HLMS) algorithm [29]. This online gradient-based approach operates through a two-step update process. The algorithm initializes the capacity μ using a uniform additive probability measure. Subsequently, for each observation $(\mathbf{x}_i, y_i)$, it conducts updates on the n coefficients of μ, which are employed to calculate

the Choquet integral $C_\mu(\mathbf{x}_i)$, by employing gradient descent to minimize the associated loss function. However, as coefficient modifications could potentially violate the monotonicity constraint, a correction step follows to ensure that μ maintains its monotonic nature. This correction procedure involves truncating updates if they conflict with adjacent coefficients.

The HLMS algorithm boasts two noteworthy advantages over the conventional quadratic program method. Primarily, it necessitates the storage of a vector representing the coefficients of μ, contrasting with the square matrix requirement of equivalent size. Secondly, the HLMS algorithm tends to converge towards less extreme solutions than the standard quadratic approach [32]. Recently, [69] have proposed a novel implementation of the HLMS algorithm utilizing automatic differentiation libraries such as PyTorch [52] and TensorFlow [1]. This approach enhances flexibility in loss function selection.

Neural networks have also been harnessed for the learning of general fuzzy measures [67]. The proposed method leverages a specific neural network architecture for computing the Choquet integral. The initial layers are dedicated to computing $K_{\mathbf{x}}(A)$ for all $A \subseteq \mathcal{N}$ without involving any learnable weights. The ultimate layer computes the inner product between the learnable weights $\mu(A)$ and $K_{\mathbf{x}}(A)$. To enforce monotonicity, the authors introduced a specially crafted soft-constraint loss function.

Another innovative neural network architecture, denoted as the ChI-NN, was introduced in [38] for the purpose of learning capacities from empirical data. Notably, the ChI-NN circumvents the need for complex monotonicity correction procedures prevalent in HLMS. This achievement is made possible by defining the fuzzy measure μ in a manner that inherently ensures monotonicity. Specifically, they establish the capacity $\mu(A)$ for any set $A \subseteq \mathcal{N}$ as the summation of the maximum value of μ over all subsets of A and an increment δ_A:

$$\mu(A) = \max_{B \subseteq A} \mu(B) + \delta_A \text{ for all } A \subseteq \mathcal{N}. \tag{2.22}$$

In this representation, the optimization variables of the learning problem are the increments δ_A for all $A \subseteq \mathcal{N}$. Enforcing non-negativity of δ_A suffices to maintain the desired monotonicity property.

A similar representation to (2.22) was previously employed in [47], where a Gibbs sampler was employed to learn the optimal capacity. Monotonicity was ensured by sampling the value of $\mu(A)$ from a distribution on the interval $[0, \min_{B \supseteq A} \mu(B)]$.

Notwithstanding these innovative heuristics, tackling problem (2.20) for moderately large universes often necessitates the introduction of additional constraints aimed at re-

ducing the number of coefficients that need to be identified or monotonicity constraints. These simplifying assumptions often manifest as k-order fuzzy measures.

The most prevalent class of k-order fuzzy measures is the category of k-additive measures. A k-additive measure is a fuzzy measure whose associated pseudo-Boolean function has a degree of at most k [30]. According to Equation (2.19), the Möbius transform of k-additive measures yields zero for all sets with a cardinality exceeding k. A formal definition for k-additive measures is presented as follows:

Definition 2.5.1. *(k-additive) A fuzzy measure μ is called k-additive ($1 \leq k \leq n$) if its Mobius transform verifies*

$$m^\mu(A) = 0$$

for all A such that $|A| > k$, and there exist a subset B with k-elements such that $m^\mu(B) \neq 0$.

For k-additive measures, the number of variables requiring learning is confined to $\sum_{i=1}^{k} \binom{n}{i}$. These measures assume no interactions among inputs of cardinality surpassing k. When addressing the fitting of k-additive measures, it is expedient to reformulate problem (2.20) with respect to its Möbius transform:

$$\begin{aligned} \min_{m^\mu} \quad & \mathcal{L}(\mu) \\ \text{s.t.} \quad & \sum_{\substack{B \subseteq A \\ i \in B \\ |B| \leq k}} m^\mu(B) \geq 0 \text{ for all } A \subseteq \mathcal{N}\ i \in A, \\ & m^\mu(\emptyset) = 0, \\ & \sum_{\substack{A \subseteq \mathcal{N} \\ |A| \leq k}} m^\mu(A) = 1. \end{aligned} \tag{2.23}$$

Problem (2.23) can be effectively tackled through Quadratic Programming (QP) or Linear Programming (LP) methodologies [9]. The instance of the 2-additive measure is often favored due to the simplification it lends to the constraints [33].

However, despite these merits, the k-additivity assumption ($k > 2$) introduces two computational challenges. Firstly, there persists a need to contend with roughly $n2^{n-1}$ monotonicity constraints. Secondly, the matrix essential for enforcing monotonicity constraints becomes densely populated.

In a quest for innovative paradigms, the concept of k-maximitive measures emerged [14]. This approach capitalizes on an alternative Möbius transform termed the possibilistic Möbius transform.

Definition 2.5.2. *(Possibilistic Möbius Transform) The possibilistic Möbius transform of a capacity μ is the set function $m^\mu : 2^{\mathcal{N}} \mapsto [0,1]$ defined by*

$$m_p^\mu(A) = \begin{cases} \mu(A) \text{ if } \mu(A) > \max_{B \subset A} \mu(B), \\ 0 \text{ otherwise.} \end{cases} \tag{2.24}$$

The k-maximitive fuzzy measure is defined analogously to the k-additive one, but employs the possibilistic Möbius transform.

Definition 2.5.3. *(k-maximitive) A fuzzy measure μ is called k-maximitive if its possibilistic Möbius transform satisfies $m_p^\mu(A) = 0$ for all $A \subseteq \mathcal{N}$, $|A| > k$ and there exists at least one subset B with k-elements such that $m_p^\mu(B) \neq 0$.*

Equivalently, a fuzzy measure is k-maxitive if $\mu(A) = \max_{B \subset A} \mu(B)$ for all $A \subseteq \mathcal{N}$ with $|A| > k$. A modification of the HLMS algorithm for k-maximitive measures was proposed by [49].

Furthermore, the concept of k-interactive measures has been introduced [10]. In this framework, values for all $A \subseteq \mathcal{N}$ with $|A| > k$ are set to maximize entropy. This formulation ensures that k-interactive fuzzy measures inherently satisfy monotonicity for sets with a cardinality surpassing k:

Definition 2.5.4. *A fuzzy measure μ on $\mathcal{N}$ is called k-interactive if for some chosen $K \in [0,1]$ and $1 \leq k \leq n$*

$$\mu(A) = K + \frac{|A| - k - 1}{n - k - 1}(1 - K) \text{ for all } A, |A| > k. \tag{2.25}$$

The value of K can be effectively estimated from data using bi-level optimization. Consequently, the Choquet integral with respect to a k-interactive measure μ is expressed as:

$$C_\mu(\mathbf{x}) = \frac{1 - K}{n - k - 1} \sum_{i=1}^{n-k-1} x_{\sigma_i} + K x_{\sigma_{n-k}} + \sum_{\substack{A \subseteq \mathcal{N} \\ |A| < k}} \mu(A) K_{\mathbf{x}}(A). \tag{2.26}$$

In this formulation, the $n - k - 1$ smallest inputs of $\mathbf{x}$ are averaged using the arithmetic mean, while interactions are considered in the context of larger inputs.

Lastly, [37] introduced the concepts of data-supported and data-unsupported variables. Data-supported variables correspond to those present in the Choquet integral formula in

the observations X, while data-unsupported variables constitute the remainder. Given that only n coefficients of μ are required for calculating the Choquet integral for a single observation, a dataset comprising m observations would entail at most mn data-supported variables (in instances where no observations are comonotone). Particularly, for moderately large n, it generally holds that $mn \ll 2^n$. This insight lends an advantageous strategy of partitioning problem (2.20) into two stages: 1) optimizing using solely the data-supported variables, and 2) resolving a feasibility problem dependent on the optimal data-supported variable values to ascertain the values of the data-unsupported variables. This strategy was effectively applied to a k-interactive linear program, enabling problem (2.20) to be addressed for cases involving up to 30 inputs [10], which, to the best of our knowledge, currently represents the state-of-the-art in this domain.

Notwithstanding years of research, the challenge of learning capacities from data in large universes ($n > 30$) persists as an ongoing concern. In the forthcoming chapter, an alternative formulation and algorithm will be introduced, exhibiting promising potential in addressing the endeavor of learning capacities from data within such contexts.

Chapter 3

Formulation and Methodology

In this chapter, we delineate our proposed methodology for the acquisition of capacities from data within large universes. Commencing with Section 3.1, we embark on the formulation of the capacity learning conundrum as an endeavor to acquire the associated pseudo-Boolean function. Notably, our exposition reveals that the utilization of polynomials as the model archetype for representing pseudo-Boolean functions aligns with the current discourse in the literature. Subsequently, we introduce an alternative model paradigm, namely the feedforward neural network, to serve as a representation for pseudo-Boolean functions, offering heightened scalability within vast universes.

The subsequent sections provide comprehensive insights into the methodology for the effective implementation of a neural network-based approach to achieve monotonic, grounded, and normalized pseudo-Boolean functions. In Section 3.2, we present a systematic protocol designed to enforce monotonicity within the neural network. By harnessing the method of multipliers, as expounded in Section 3.3, we introduce an inventive technique to ensure the neural network's capability to estimate a grounded and normalized pseudo-Boolean function. Advancing further, Section 3.4 unveils an algorithm designed to efficiently compute the Choquet integral in scenarios characterized by high dimensionality.

Concluding our methodology exposition, Section 3.5 offers insights into the current configuration of hyper-parameters. Throughout this chapter, we shall diligently adhere to the notation previously established in Chapter 2.

3.1 A More General Formulation

As elucidated in Section 2.3, a direct correspondence exists between monotone, grounded, and normalized pseudo-Boolean functions and fuzzy measures. Consequently, we can reframe the challenge of learning the optimal capacity from data as an endeavor to learn the associated pseudo-Boolean function:

$$\begin{aligned} \min_{f_\mu \in \mathcal{PB}(n)} \quad & \mathcal{L}(\mu) \\ \text{s.t.} \quad & f_\mu \text{ is monotone }, \\ & f_\mu(\mathbf{0}) = 0, \\ & f_\mu(\mathbf{1}) = 1, \\ & \mu(A) = f_\mu(\mathbf{1_A}) \; \forall A \subseteq \mathcal{N}. \end{aligned} \tag{3.1}$$

Here, $\mathcal{PB}(n)$ denotes the set encompassing all pseudo-Boolean functions defined on $\{0,1\}^n$. To address the optimization problem (3.1) in practice, we need to impose a constraint that the pseudo-Boolean functions originate from a model class $\mathcal{PB}^\theta(n) = \{f_\theta : \{0,1\}^n \mapsto \mathbb{R} | \theta \in \mathbb{R}^p\} \subseteq \mathcal{PB}(n)$ and then perform the optimization over its parameters.

This formulation leads us to the following reformulation:

$$\begin{aligned} \min_{\theta \in \mathbb{R}^p} \quad & \mathcal{L}(\mu) \\ \text{s.t.} \quad & f_\theta \in \mathcal{PB}^\theta(n), \\ & f_\theta \text{ is monotone }, \\ & f_\theta(\mathbf{0}) = 0, \\ & f_\theta(\mathbf{1}) = 1, \\ & \mu(A) = f_\theta(\mathbf{1_A}) \; \forall A \subseteq \mathcal{N}. \end{aligned} \tag{3.2}$$

The choice of a suitable model greatly influences our research direction. Unlike the established standard and Möbius representations, which lead to the challenges outlined in problem (2.20), we introduce an innovative approach. Our unique perspective involves adopting an alternative model class, moving away from reliance on polynomials. Specifically, we explore $\mathcal{PB}^\theta(n) = \{\hat{f}_\theta : \{0,1\}^n \mapsto \mathbb{R} \mid \theta \in \mathbb{R}^p\}$, where $\hat{f}_\theta$ denotes a feed-forward neural network.

This network follows a defined architecture:

$$z_{i+1} = \gamma_i(W_i z_i + b_i). \tag{3.3}$$

For $i = 1, ..., l$, the representation proceeds as follows: z_i is the activation at layer i, starting with $z_0 = \mathbf{x}$, and γ_i represents non-linear activation functions. The matrix $W_i \in \mathbb{R}^{h_{i+1} \times h_i}$ holds the neural network weights, and $b_i \in \mathbb{R}^{h_{i+1}}$ is for biases. The collection of all weights and biases forms the parameter $\theta = \{W_i, b_i\}$. The depth of the neural network is labeled as l, and h_i indicates the size of the hidden layer.

This approach opens a realm of possibilities as it exhibits the potential to circumvent several challenges encountered when employing standard and Möbius representations in the context of large universes. The upcoming sections will thoroughly address the methodological challenge of effectively integrating feed-forward neural networks into this innovative framework.

3.2 Enforcing Monotonicity

The pursuit of enforcing monotonicity within deep neural networks has emerged as a vibrant research domain. Existing methodologies within the literature can be broadly categorized into two distinctive classes:

1. Monotonicity by Construction: This category encompasses neural architectures that inherently ensure monotonicity through their design and construction. These architectures are explicitly crafted to maintain monotonic relationships within the model's output [4, 60, 21, 15, 71].

2. Monotonicity by Regularization: In this approach, the objective is to enforce monotonic behavior during the training process. This is achieved by incorporating alterations to the loss function or integrating heuristic regularization terms that encourage monotonicity in the learned model [61, 36, 44].

The simplest approach to ensure the monotonicity of a standard feed-forward neural network by construction is by enforcing non-negativity of the weights alongside the utilization of a monotonic activation function. Consider a standard feed-forward neural network with three layers, characterized as follows:

- An input layer housing k nodes.
- A hidden layer featuring h nodes.
- An output layer comprising one or more nodes.

All connections between these layers are weighted. Denote the weight between input j and hidden node i as w_{ij}, and let v_i represent the weight between the output and hidden node i. Given an input $\mathbf{x} \in \mathbb{R}^k$, the functional form of the output $O(\mathbf{x})$ corresponding to a single hidden layer neural network is expressed as:

$$O(\mathbf{x}) = \gamma \left(\sum_{i=1}^{h} v_i \gamma \left(\sum_{j=1}^{k} w_{ij} x_j + b_i \right) + b_0 \right), \tag{3.4}$$

where b_0, b_i $(i = 1, ..., h)$ denote the bias terms, and γ signifies the activation function.

To preserve monotonicity in the output concerning input x_j, it is imperative that the partial derivative of the output $O(x)$ with respect to x_j remains non-negative:

$$\frac{\partial O(x)}{\partial x_j} = \gamma' \sum_{i=1}^{h} v_i \gamma' \left(\sum_{j=1}^{k} w_{ij} x_j + \theta_i \right) w_{ij} \geq 0. \tag{3.5}$$

When $\gamma' > 0$, equation (3.5) is satisfied if and only if all weights are non-negative:

$$\forall 1 \leq i \leq h, \text{ and } 1 \leq j \leq k, \quad v_i w_{ij} \geq 0. \tag{3.6}$$

In this manner, by enforcing non-negativity of weights and incorporating a monotonic activation function, the standard feed-forward neural network is inherently designed to maintain monotonic relationships between its inputs and outputs.

In addition, neural networks with non-negative weights can approximate any continuous function defined on a compact subset arbitrarily well. This remarkable capability is formally stated through the following theorem:

Theorem 3.2.1. *For any continuous monotone increasing function $f : K \mapsto \mathbb{R}$, with K a compact subset of $\mathbb{R}^k$, there exist a feedforward neural network with at most k hidden layers, non-negative weights, and output O such that $|f(\mathbf{x}) - O(\mathbf{x})| \leq \epsilon$ for any $\mathbf{x} \in K$ and $\epsilon > 0$.*

Proof. Refer to Theorem 3.1 in [21]. □

It's essential to acknowledge that Theorem 3.2.1 exclusively holds true for saturated activation functions.

Definition 3.2.1. *(Saturated function) A function $\phi : \mathbb{R} \mapsto \mathbb{R}$ is classified as left-saturated if $\lim_{x\to-\infty} \phi^{'}(x) = 0$, and it is termed right-saturated if $\lim_{x\to\infty} \phi^{'}(x) = 0$, where $\phi^{'}$ signifies the derivative of ϕ. A function is deemed saturated if it satisfies both left-saturation and right-saturation conditions.*

However, an inherent issue arises from this context, as deep neural networks with saturated activation functions are notoriously challenging to train due to the predicament of vanishing gradients.

Conversely, the triumph of well-established activation functions in deep learning, such as ReLU [51], ELU [18], SeLU [43], etc., are monotonically increasing convex functions. Within the framework of a fully-connected feed-forward neural network, containing at least one hidden layer, and with unconstrained weights, these monotonically increasing convex activation functions can approximate any continuous functions over compact domains.

Nonetheless, the introduction of non-negativity constraints on weights poses a substantial limitation. When weights are restricted to be non-negative, the resulting neural network is incapable of approximating non-convex functions.

Theorem 3.2.2. *Let $O(\mathbf{x})$ denote the output of a neural network featuring non-negative weights and a monotonically increasing convex activation function. In such a scenario, O exhibits convexity in $\mathbf{x}$.*

Proof. The proof is straightforward and stems directly from the principle that non-negative sums of convex functions also yield convex functions (as detailed in section 3.2.4 of [13]). For a more comprehensive explanation, refer to the proof of proposition 1 in [3]. □

Given these limitations, the quest for alternative activation functions becomes imperative. In this study, we adopt the activation function $\rho^s : \mathbb{R}^h \mapsto \mathbb{R}^h$, introduced in [57], which not only adheres to Theorem 3.2.1, ensuring robust approximations, but also furnishes advantageous gradient properties within deep architectures.

Central to their construction is the amalgamation of three distinct monotonically increasing functions, with each function applied to specific subsets of neurons within a given layer:

- A monotonically increasing, convex, and lower-bounded function denoted as $\breve{\rho}$.
- A concave, upper-bounded function referred to as $\hat{\rho}$.
- A saturated activation function designated as $\tilde{\rho}$.

Definition 3.2.2. *Consider $\breve{\rho} : \mathbb{R} \mapsto \mathbb{R}$ as a monotonically increasing, convex, and lower-bounded function. By introducing a concave upper-bounded activation function, we define:*

$$\hat{\rho}(x) = -\breve{\rho}(-x). \tag{3.7}$$

Additionally, a saturated activation function $\tilde{\rho}$ is outlined as:

$$\tilde{\rho}(x) = \begin{cases} \breve{\rho}(x+1) - \breve{\rho}(1), & \text{if } x < 0, \\ \hat{\rho}(x-1) + \hat{\rho}(1), & \text{otherwise.} \end{cases} \tag{3.8}$$

The activation function's parametrization relies on activation weights $\mathbf{s} = (s_1, s_2, s_3) \in \mathbb{N}^3$ where $s_1 + s_2 + s_3 = h$.

Definition 3.2.3. *Given activation weights $\mathbf{s} = (s_1, s_2, s_3) \in \mathbb{N}^3$, subject to $s_1+s_2+s_3 = h$, the activation function $a^s : \mathbb{R}^h \mapsto \mathbb{R}^h$ is explicitly defined as follows:*

$$\rho^s(\mathbf{x}) = \begin{cases} \breve{\rho}(x_j), & \text{if } j \leq s_1, \\ \hat{\rho}(x_j), & \text{if } s_1 < j \leq s_1 + s_2, \\ \tilde{\rho}(x_j), & \text{otherwise.} \end{cases} \tag{3.9}$$

This carefully constructed activation function ρ^s amalgamates the properties of the individual functions, and its application in neural networks with non-negative weights offers a solution to both the challenge of theoretical guarantees of monotonicity and the issue of vanishing gradients often associated with saturated activation functions.

Moving forward, we will adopt the notation $\Theta = \{\theta \in \mathbb{R}^p : \hat{f}_\theta \text{ is monotone}\}$ to signify the parameter set that guarantees the monotonicity of the deep neural network. In accordance with our chosen architectural design, Θ takes on the form $\Theta = \{\theta \in \mathbb{R}^p \,|\, W_i \geq 0, i = 1, ..., l\}$.

By leveraging the ease of projection onto the polytope Θ, we can reframe the original problem (3.2) into a significantly simplified equality-constrained optimization problem:

$$\begin{aligned} \min_{\theta \in \Theta} \quad & \mathcal{L}(\theta) \\ \text{s.t.} \quad & \mathcal{C}(\theta) = \mathbf{0}. \end{aligned} \tag{3.10}$$

Within this context, $\mathcal{C}(\theta) : \mathbb{R}^p \mapsto \mathbb{R}^2$ computes the extent of violation of the grounded and normalized constraints:

$$\mathcal{C}(\theta) = \begin{bmatrix} \hat{f}_\theta(\mathbf{0}) \\ \hat{f}_\theta(\mathbf{1}) - 1 \end{bmatrix}. \tag{3.11}$$

In the subsequent section, we elucidate our methodology for addressing and resolving the optimization problem outlined in (3.10).

Remark 3.2.1. *(Universal approximator) As stipulated in Theorem 3.2.1, the deep neural network's approximation of a continuous monotone function necessitates k hidden layers. It is noteworthy that, in general, a one hidden layer neural network endowed with non-negative weights falls short in approximating multivariate continuous monotone functions. A concrete counter-example within the context of two dimensions can be found in Appendix I of [21].*

Nevertheless, our specific focus centers around the approximation of pseudo-boolean functions. Intriguingly, we propose a conjecture that a one hidden layer neural network boasting 2^n neurons could potentially suffice in approximating all monotone pseudo-boolean functions. Our conjecture is rooted in the observation that the polynomials (2.15) constitute a basis encompassing 2^n dimensions within the vector space $\mathcal{PB}(n)$.

Regrettably, our attempts to furnish a formal proof have not yielded success. We encourage readers intrigued by this prospect to delve into [25], which offers a proof that a one hidden layer max-min neural network incorporating $2^n - 1$ neurons (sans constraints on the weights) can aptly approximate all pseudo-boolean functions. Furthermore, [20] furnishes evidence that a pseudo boolean function is monotone if and only if it aligns with a pseudo-polynomial function.

Remark 3.2.2. *(Non-negative weights) To enforce the requirement of non-negative weights, we adopt a strategy proposed in [57] that involves taking the absolute value of unconstrained weights.*

Concretely, in the architecture of our deep neural network, each layer is characterized by the functional structure:

$$\mathbf{y} = a^s(|W|\mathbf{x} + \mathbf{b}). \tag{3.12}$$

In this context, the components assume the following significance:

- $\mathbf{x} \in \mathbb{R}^k$ *represents the input vector;*

- $\mathbf{y} \in \mathbb{R}^h$ *denotes the output vector;*
- $W \in \mathbb{R}^{k \times h}$ *represents the matrix of learnable weights;*
- $\mathbf{b} \in \mathbb{R}^h$ *signifies the learnable bias terms.*

The operation $|W|$ *corresponds to element-wise application of the absolute value to each entry in* W*. In explicit terms, if* $M = |W|$*, then* $m_{ij} = |w_{ij}|$*.*

3.3 Enforcing Boundary Conditions

Stochastic Gradient Descent (SGD) [41, 56] and its variants [42] are the standard method of training neural networks. It serves as the bedrock of the training process, iterating through the dataset with randomized subsets to efficiently navigate the vast parameter space. However, it is essential to note that SGD's applicability remains confined to unconstrained minimization problems, where the optimization landscape lacks explicit constraints. In instances, such as problem (3.10), where we have constraints we must transform the problem to a sequence of unconstrained optimization problems. There exist three common methods to convert hard equality constrained optimization problems to an unconstrained optimization problem: 1) the penalty method, 2) primal-dual methods, and 3) the method of multipliers.

The penalty method transforms an equality constrained optimization problem into a sequence of unconstrained minimization problems, achieved by systematically increasing the penalty parameter within the objective function

$$\min \mathcal{L}(\theta) + c_k \frac{||\mathcal{C}(\theta)||_2^2}{2}, \tag{3.13}$$

with $c_k \leq c_{k+1}$ [27]. Under mild assumptions, the penalty method is guaranteed to converge to a local minimum in the limit that $c_k \to \infty$ [27]. The drawback of the penalty method is that it can encounter numerical instability particularly when the penalty parameter c assumes substantial values.

Primal-dual methods are founded on the sequential minimization of the Lagrangian function:

$$L(\theta, \lambda) = \mathcal{L}(\theta) + \lambda^T \mathcal{C}(\theta). \tag{3.14}$$

In its simplest form, one minimizes $L(\theta, \lambda_k)$ for a sequence of dual variables λ_k [45]. This sequence is generated by the update rule:

$$\lambda_{k+1} \leftarrow \lambda_k + \alpha_k \mathcal{C}(\theta_k), \tag{3.15}$$

where θ_k denotes the minimizing point of $L(\theta, \lambda_k)$ over Θ, and α_k is the step size. There are two significant disadvantages associated with utilizing the primal-dual method. Firstly, the problem defined in equation (3.10) must exhibit a locally convex structure to ensure the well-defined nature of the dual problem and the meaningfulness of iteration (3.15). Secondly, the convergence rate of the method is slow.

The method of multipliers seamlessly integrates the penalty method with the primal-dual philosophy. In this method, a penalty term is incorporated into the Lagrangian function to construct the augmented Lagrangian function:

$$L_c(\theta, \lambda) = \mathcal{L}(\theta) + \lambda^T h(\theta) + c\frac{||\mathcal{C}(\theta)||_2^2}{2}. \tag{3.16}$$

The method of multipliers solves a sequence of unconstrained minimization problems as outlined in Algorithm 1. It boasts three key advantages [11]:

1. Convergence can be achieved without the need to escalate c_k to infinity, mitigating the numerical stability challenges commonly associated with the penalty method.

2. The method of multipliers exhibits rapid convergence towards the Lagrange multiplier vector.

3. The algorithm's convergence is not reliant on problem (3.10) possessing a locally convex structure.

As a result, in this research project we use the method of multipliers to convert problem (3.10) to a sequence of unconstrained minimization problems.

The method of multipliers has previously been employed to address constrained optimization challenges within the realm of deep neural networks. For instance, in [23], the method of multipliers was harnessed to enforce boundary conditions in a physics-informed neural network (PINN), while [58] utilized this approach to enhance classification performance in scenarios characterized by class imbalance.

However, applying Algorithm 1 directly to a deep neural network necessitates certain adaptations. As previously mentioned, we employ Stochastic Gradient Descent (SGD) for the inner optimization step, denoted by $\theta_{k+1} \leftarrow \arg\min_{\theta \in \Theta} L_{c_k}(\theta, \lambda_k)$.

Hyper-parameters: initial penalty parameter c_0, factor $\beta > 1$, tolerance ϵ

$k \leftarrow 0$;

$\lambda_0 \leftarrow \mathbf{0}$;

while $||\mathcal{C}(\theta_k)||_2 > \epsilon$ **do**

  $\theta_{k+1} \leftarrow \arg\min_{\theta \in \Theta} L_{c_k}(\theta, \lambda_k)$;

  $\lambda_{k+1} \leftarrow \lambda_k + c_k h(\theta_{k+1})$;

  $c_{k+1} \leftarrow \beta c_k$;

  $k \leftarrow k + 1$;

end

Algorithm 1: Method of Multipliers

The initial challenge posed by SGD lies in determining convergence, as conventional criteria like gradient norm falling below a tolerance are inapplicable. This makes deciding when to terminate the inner optimization a nontrivial task. To address this, we resort to arguably the simplest heuristic – conducting optimization for a predetermined number of epochs denoted as T. Here, an epoch entails a complete dataset enumeration.

Another challenge arises when employing optimizers like Adam [42] to determine the learning rate. While Adam is the common choice for training deep learning models and often outperforms basic stochastic gradient descent in unconstrained optimization problems, complexities arise when dealing with the method of multipliers. Adjusting the penalty parameter c and the Lagrangian vector λ with each update changes the training subproblem. Consequently, the learning rate adaptation strategies used by the Adam optimizer in the previous optimization problem can impede the convergence of the current optimization problem. Drawing inspiration from [23], we opt to reset the optimizer whenever we modify the penalty factor c_k and the Lagrangian vector λ_k, treating each updated problem as an independent instance.

Algorithm 2 shows the Method of Multipliers to train a deep neural network.

Remark 3.3.1. *(Fixed Penalty Method) The fixed penalty method enhances the original objective function by incorporating a penalty term that quantifies the extent of constraint violation:*

$$\min \mathcal{L}(\theta) + c\frac{||\mathcal{C}(\theta)||_2^2}{2}. \tag{3.17}$$

Here, c denotes the penalty parameter. This approach notably simplifies the transformation of a constrained problem into an unconstrained one, representing a straightforward

Hyper-parameters: initial penalty parameter c_0, maximum penalty parameter c_{max}, factor $\beta > 1$, number of epochs for the inner optimization T, tolerance ϵ

$k \leftarrow 0$;
$\lambda_0 \leftarrow \mathbf{0}$;
Initialize θ_0^0 randomly ;
while $||\mathcal{C}(\theta_0^k)||_2 > \epsilon$ **do**
    for *t=1,...,T* **do**
        for *each mini-batch of* X_b *with size* B **do**
            $y_b = C_\mu(X_b)$; `/* Compute the Choquet Integral using Algorithm 3 */`
            Udpate θ_{t+1}^k using the Adam optimizer ;
        end
    end
    $\theta_0^{k+1} \leftarrow \theta_T^k$;
    $\lambda_{k+1} \leftarrow \lambda_k + c_k \mathcal{C}(\theta_T^k)$;
    $c_{k+1} \leftarrow \min(c_{max}, \beta c_k)$;
    $k \leftarrow k + 1$;
    Reinitialize the Adam optimizer ;
end

Algorithm 2: Method of Multipliers for training deep neural networks

means of handling constraints.

It stands as one of the most prevalent techniques for enforcing equality constraints in the realm of deep learning, as evidenced by its adoption in various studies [48, 22, 39]. However, it falls short in providing strict adherence to the constraints. The introduction of the penalty term introduces a delicate balance between the objective and constraint fulfillment, necessitating meticulous tuning of the penalty parameter c.

In instances where a larger degree of tolerance towards deviations in boundary conditions is acceptable, the fixed penalty method may present a more suitable alternative than the method of multipliers for addressing problem (3.10).

3.4 Computing the Choquet Integral

The final methodological challenge we need to tackle pertains to the computation of the Choquet integral, an essential requirement for evaluating the loss function. Numerous mathematical program formulations [9] and data-driven algorithms for learning fuzzy measures [38] employ equation (2.11) to calculate the Choquet integral. This equation's utility stems from its ability to represent the capacity μ values for all $A \subseteq \mathcal{N}$ through an array $v \in [0,1]^{2^n}$, and similarly, to encapsulate $K_{\mathbf{x}}(A)$ values for all $A \subseteq \mathcal{N}$ within an array $\mathbf{K}_{\mathbf{x}} \in \mathbb{R}^{2^n}$. Consequently, the Choquet integral of $\mathbf{x}$ relative to μ equates to the inner product of these two arrays.

An inherent drawback of using equation (3.18) lies in its exponential time complexity with respect to n. When employing Choquet-integral based models on large universes, the selection of algorithms becomes crucial to ensure polynomial-time evaluation of the Choquet integral in relation to n. Algorithm 3, based on equation (2.9), aligns with this requirement and addresses our imperative [9].

The algorithm necessitates sorting the input $\mathbf{x}$, incurring a computational expense of $\mathcal{O}(n \log n)$, and involves n function calls to the neural network. Assuming that we possess the ability to evaluate the neural network in polynomial time concerning n, it follows that we can compute the Choquet integral with respect to μ within a polynomial timeframe relative to n.

In practical applications, we further enhance computation speed by evaluating inputs in batches of size B, capitalizing on well-optimized tensor operations. Consider the matrix $X_B = [\mathbf{x}^1; ...; \mathbf{x}^B] \in [0,1]^{B \times n}$, where each row corresponds to an input within the batch. We adapt Algorithm 3 to efficiently evaluate batches of inputs through the following steps:

Input: f_μ pseudo-boolean function associated with the capacity μ, $\mathbf{x} \in [0,1]^n$
Output: $C_\mu(\mathbf{x})$ the Choquet integral evaluated at $\mathbf{x}$ with respect to μ

$\sigma \leftarrow \text{argsort}(\mathbf{x})$; `/* Permutation of x s.t.` $x_{\sigma_1} \leq ... \leq x_{\sigma_n}$ `*/`
Let $x_{\sigma_0} = 0$;
$z \leftarrow \mathbf{1}$ and $\mathcal{S} \leftarrow 0$;
for *j=1 to n* **do**
  $\mathcal{S} \leftarrow \mathcal{S} + [x_{\sigma_j} - x_{\sigma_{j-1}}] f_\mu(z)$;
  $z_{\sigma_j} \leftarrow 0$
end
Return $\mathcal{S}$

Algorithm 3: Computing the Choquet Integral of a pseudo-Boolean function

1. Initially, we employ a loop to compute a matrix $Z \in \{0,1\}^{Bn \times n}$, encoding the Boolean values necessary for evaluating the Choquet integral across all inputs in X_B.

2. Subsequently, we acquire the coefficient values of the capacity by evaluating $V = \hat{f}(Z) \in [0,1]^{Bn}$ and reshape V to form a matrix of dimensions $B \times n$.

3. The third step involves the computation of a matrix $S \in [0,1]^{B \times n}$, with each entry following the pattern $S_{ij} = \mathbf{x}^i_{\sigma^i_j} - \mathbf{x}^i_{\sigma^i_{j-1}}$, mirroring the approach outlined in Algorithm 3. Here, $\mathbf{x}^i_{\nearrow} = (x^i_{\sigma^i_1}, x^i_{\sigma^i_2}, ..., x^i_{\sigma^i_n})$ is a non-decreasing permutation of the input $\mathbf{x}^i$, while $x_{\sigma^i_0} := 0$.

4. Finally, the Choquet integral of X_B is deduced from the diagonal elements of the matrix product between V and S, yielding $C_\mu(X_B) = \text{diag}(VS)$.

Remark 3.4.1. *(Batching Comonotone Inputs Together) A potential strategy to achieve further acceleration in computing the Choquet integral involves grouping comonotone inputs into batches. In scenarios where inputs exhibit comonotonic behavior, a mere n function calls to the neural network would suffice, contrasting with the requirement of Bn function calls for evaluating the Choquet integral across the entire batch of inputs. The extent of this efficiency gain hinges on the quantity of inputs in the dataset that manifest comonotone attributes.*

Remark 3.4.2. *(Alternative Formulation of the Choquet Integral) Perceptive readers may have observed that in Algorithm 3, we employed the following equation for the computation*

of the Choquet integral:

$$C_\mu(x) = \sum_{i=1}^{n} \mu(H_i) \left(x_{\sigma_i} - x_{\sigma_{i-1}} \right). \tag{3.18}$$

Under the assumption that $\mu(\emptyset) = 0$*, equation* (3.18) *essentially represents a rearrangement of terms present in equation* (2.9)*. However, practical considerations come into play, particularly in the initial stages of the training process, where* $\hat{f}_\theta(\mathbf{0}) \neq 0$*. This difference in initial values will lead to distinct Choquet integral outcomes between the two equations. It should be noted that we have not yet conducted tests to determine whether utilizing equation* (2.9) *in lieu of equation* (3.18) *influences the algorithm's convergence dynamics. Further investigation is required to establish any potential impact on algorithmic convergence.*

3.5 Hyper-parameters

Our approach involves several hyper-parameters that influence our results. To ensure the reproducibility of our experiments, we have documented the specific values of these hyper-parameters, as shown in Table 3.1. These values were determined through a process of iterative experimentation and by drawing inspiration from successful settings in prior research papers.

While the hyper-parameters we have chosen have produced promising outcomes, it is important to note that there might exist even better settings. Future research could explore the use of Bayesian optimization techniques, such as those outlined in [2], to fine-tune these hyper-parameters and potentially uncover improved configurations.

Furthermore, it is noteworthy that we used the default parameters as implemented in the PyTorch library [52] for the Adam optimizer.

Name	Symbol	Value
batch size	B	10
initial penalty parameter	c_0	1e-3
tolerance	ϵ	1e-4
number of epochs	T	100
factor	β	2
maximum penalty parameter	$c_{\max}$	1e6
activation weights	$\mathbf{s}$	$s_1 = \lfloor \frac{7h}{14} \rfloor, s_2 = \lfloor \frac{7h}{14} \rfloor, s_3 = h - s_1 - s_2$
convex activation function	$\breve{p}$	ReLU

Table 3.1: Values of Hyper-parameters

Chapter 4

Results

In this chapter, we present a comprehensive account of our numerical experiments, which were conducted to evaluate the efficiency (section 4.1) and accuracy (section 4.2) of our proposed method.

We use the term Pseudo-Boolean Neural Network (PB-NN) to succinctly denote the approach outlined in Chapter 3 throughout this chapter. In our comparative analysis, we subject PB-NN to benchmarking against two notable methods: ChI-NN [38] and the k-interactive linear program [10].

Our selection of ChI-NN as a benchmark is grounded in the fact that it represents a previous effort employing a Deep Neural Network for learning capacities from data. In parallel, we adopt the k-interactive linear program as another benchmark, recognizing its unique capacity to handle large universes. The value of k for the k-interactive linear program is set as the minimum between the input size n and 5, and we implement this method using the fittingKinteractiveAuto function from the Rfmtools package,[1] which harnesses bi-level optimization to optimally determine the value of K.

To ensure a robust assessment, we adhere to the original implementation[2] and default hyper-parameter settings recommended by the authors for ChI-NN.

[1] https://cran.r-project.org/web/packages/Rfmtool/index.html
[2] https://github.com/aminb99/choquet-integral-NN

4.1 Computation Time

Our initial experiment aims to investigate the impact of varying universe sizes on the runtime performance of PB-NN. In this experiment, we maintain the hyper-parameters as detailed in section 3.5, and we opt for a neural architecture featuring two hidden layers, each comprising 32 nodes.

To comprehensively explore the effects of universe size, we examine scenarios across a range of n values, specifically ranging from $n = 3$ to $n = 15$. For each specific universe size, we generate a dataset encompassing 100 instances. These instances are drawn from a uniform distribution, thereby creating the feature matrix $X \in [0, 1]^{100 \times n}$.

For the target values $\mathbf{y} \in [0, 1]^{100}$, we employ the Choquet integral with respect to a capacity μ, which is generated randomly. To facilitate this generation process, we employ the generate_fm_minplus function from the Rfmtools package. Key parameters for this function are set as follows: *kint* is assigned the value of n, *markov* is set to 1000, and *option* is set to 1. It's worth noting that the approach for random capacity generation is elucidated in [19].

Figure 4.1 provides a visual representation of the impact of the input size n on the run-time performance of three methods: PB-NN, the k-interactive linear program, and ChI-NN.

In instances involving small universes, both ChI-NN and the k-interactive linear program outperform PB-NN significantly. For instance, when considering $n = 3$, PB-NN's convergence took nearly a minute, contrasting with the ChI-NN and k-interactive linear program which achieved convergence within a matter of seconds.

Nevertheless, a discernible trend emerges as the universe size n grows. While both ChI-NN and the k-interactive linear program exhibit significant increases in run-time, PB-NN continues to display efficient performance that scales adeptly with larger n values. Notably, at $n = 15$, PB-NN surpasses the performance of both ChI-NN and the k-interactive linear program. This finding underscores the robust computational scalability of PB-NN concerning universe size, positioning it as a promising choice for addressing large problems.

It is intriguing to note that the run-time of the k-interactive linear program experiences a drop when transitioning from a universe size of $n = 9$ to $n = 10$. While the exact cause of this phenomenon is yet to be determined, further investigation is warranted to gain a comprehensive understanding. Importantly, we maintain the perspective that despite such anomalies, the overarching trend remains intact. As n continues to increase, the run-time of the k-interactive linear program rises significantly, ultimately rendering it impractical for handling very large universes.

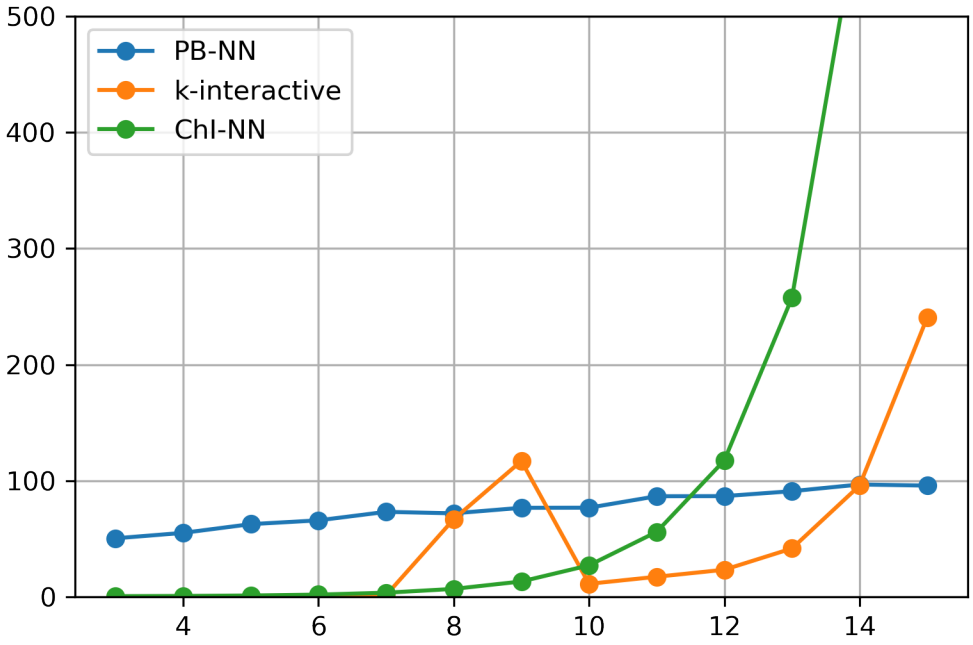


Figure 4.1: CPU run time for different input sizes

It is noteworthy that our run-time results for the k-interactive linear program align with those reported in [10]. This alignment reinforces the consistency of our findings and lends credibility to our observations regarding the scalability limitations of the k-interactive linear program in comparison to PB-NN.

In order to further emphasize the scalability of PB-NN, an additional experiment is conducted within larger universes. Specifically, we consider universe sizes of $n = 20, 50, 80,$ and 100. Given the computational challenges of generating random capacities within these expansive universes, an alternative approach is employed.

Initially, 100 samples are randomly selected from the diabetes dataset.[3] The target values $\mathbf{y}$ are then transformed to adhere to a uniform distribution through the use of the empirical cumulative distribution.[4] To derive the feature matrix, n trees are fitted to the

[3]https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html
[4]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html

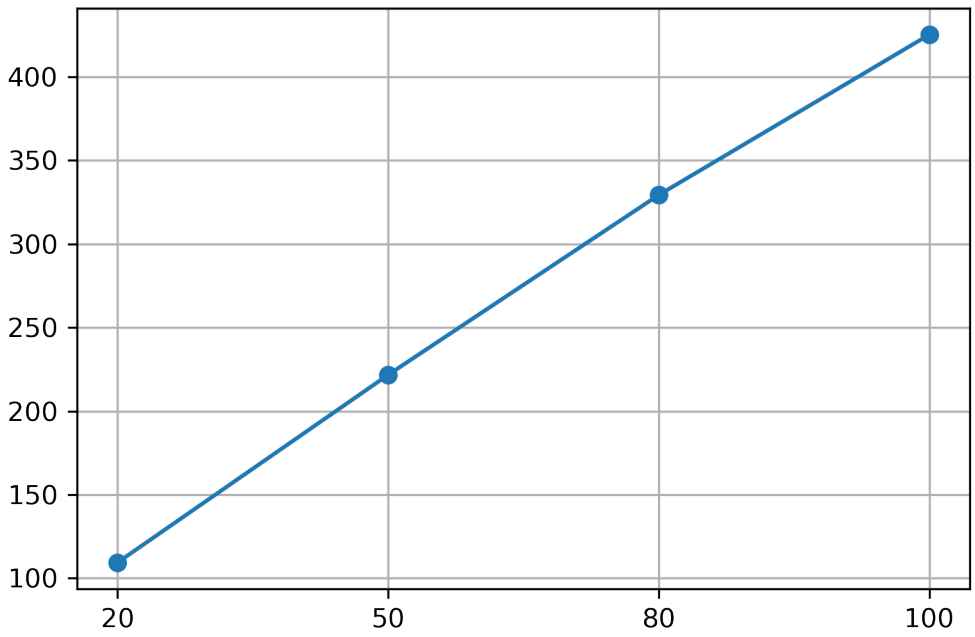


Figure 4.2: CPU run time for the PB-NN in large universes

diabetes dataset using a random forest,[5] and the predictions from each tree serve as our features.

Figure 4.2 offers insight into the average run time of PB-NN across five distinct trials for these larger universes. Notably, the algorithm's run time does not exhibit exponential growth concerning the input size n, thereby reinforcing the ability of PB-NN to effectively learn capacities from data within substantial universes.

Of particular interest is the observation that PB-NN can solve problems within a universe featuring an input size of 100 in just over 7 minutes. This accomplishment stands out as PB-NN is, to the best of current knowledge, the pioneering methodology capable of efficiently learning from data within universes of such a scale.

Remark 4.1.1. *(Effect of sample size) In our experimental observations, we identified a noteworthy trend regarding the influence of sample size on PB-NN's run-time. Specifically,*

[5] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

we observed that the run-time of PB-NN exhibits a linear relationship with the sample size m. Doubling the sample size approximately results in a doubling of the run-time as well. This behavior aligns with our expectations, given that each epoch entails a complete iteration through the dataset.

Conversely, the impact of sample size on the run-time of a linear program differs. For a linear program, the run-time is comparatively less affected by an increase in sample size. This discrepancy is rooted in the fact that, for m samples, a linear program necessitates the inclusion of $2m$ linear constraints. In the context of a moderately large n, this constraint count is typically minor in comparison to the number of constraints required to enforce monotonicity.

Remark 4.1.2. *(Computer hardware) All of our computational experiments were conducted on a 2020 MacBook Air, which is equipped with 8 GB of memory. It's important to acknowledge that by using this specific hardware, we have not fully leveraged the extensive advancements in deep neural network training optimization that have emerged in recent years.*

We anticipate that the performance of PB-NN could be further enhanced when executed on computer hardware that supports parallel computing, such as shared memory systems, distributed memory systems, or GPUs. These hardware configurations can capitalize on the cutting-edge techniques and methodologies that have been developed to accelerate the training of deep neural networks. Consequently, we expect that utilizing parallel computing capabilities will enable us to tackle larger problems or problems of comparable size more efficiently with PB-NN.

4.2 Model Accuracy

The objective of our second experiment is to assess the PB-NN's ability to accurately learn capacities from data. To address this, we design a synthetic experiment wherein the true capacity μ is known. The experiment's procedure is detailed as follows:

1. Generate 10 capacities randomly, denoted as μ_i for $i = 1, ..., 10$, using the method described in [19].

2. Create a training set, referred to as X_{train}, by drawing 100 samples from a uniform distribution. Generate a separate test set, denoted as X_{test}, containing 10,000 samples.

3. Compute the training targets $\mathbf{y}^i_{train}$ by calculating the Choquet integral at X_{train} with respect to μ_i. Similarly, determine the test targets $\mathbf{y}^i_{test}$ by computing the Choquet integral at X_{test} with respect to μ_i.

4. Introduce varying levels of noise to the training set. Define σ^i_y as the standard deviation of the training target $\mathbf{y}^i_{train}$. Add noise sampled from a normal distribution with a mean of 0 and a standard deviation of 0, $0.1\sigma^i_y$, and $0.5\sigma^i_y$ to the training target $\mathbf{y}^i_{train}$ creating the noiseless, noisy, and very noisy cases, respectively. The test targets $\mathbf{y}_{test}$ always remain noiseless.

5. Utilize two distinct neural architectures for experimentation: PB-NN5 and PB-NN32. Both architectures feature two hidden layers, with the former having 5 nodes per layer and the latter having 32 nodes per layer.

6. Repeat the entire experiment for three different universe sizes: $n = 3, 8$, and 15.

A core aim of this comprehensive experiment is to thoroughly evaluate PB-NN's performance across a spectrum of noise levels, diverse neural architectures, and various universe sizes. To ensure a fair and equitable comparison with the k-interactive linear program, we fit all algorithms within the framework of the least absolute deviation.

To assess the effectiveness of the algorithms, we employ three evaluation metrics. Firstly, we gauge the mean absolute deviation between the predicted capacity from the algorithms and the coefficients of the true capacity. Additionally, we compute the coefficient of determination (R^2) for both the training and test datasets. We utilize the r2_score function from the scikit-learn library[6] to calculate the coefficient of determination. The coefficient of determination R^2 between the actual labels $\mathbf{y}$ and the predicted labels $\hat{\mathbf{y}}$ is defined as follows:

$$R^2(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y})^2}, \tag{4.1}$$

where $\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y_i$ is the expected value of the target labels $\mathbf{y}$. The R^2 value can be negative when the model's predictions are worse than those of a constant model that always predicts the expected value of $\mathbf{y}$. While there is no direct mapping between the coefficient of determination and our l_1 norm loss function, they share a robust inverse relationship. Improvements in R^2 generally correspond to reduced l_1 loss. Due to this connection and the intuitive interpretability of R^2 as a measure of the goodness of fit, we have selected it as

[6] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html

our evaluation metric to assess the performance of PB-NN in accurately learning capacities from data in our experimental setting.

4.2.1 3 Inputs

In the subsequent section, we present the outcomes of our experiment conducted in a small universe comprising 3 inputs. It's important to note that both PB-NN5 and PB-NN32 encompass a significantly higher number of learnable parameters—56 and 1217 respectively—compared to the 8 parameters required to define the capacity. The ChI-NN consistently maintains the same quantity of parameters as those needed for capacity identification, which amounts to 2^n. Consequently, in this scenario, the ChI-NN is equipped with 8 learnable parameters.

Tables 4.1, 4.2, and 4.3 present a comprehensive overview of the true capacity coefficients, as well as the predicted coefficients by all algorithms, for a selected experiment under noiseless, noisy, and very noisy conditions, respectively. These tables exhibit the instance where PB-NN32 exhibited the largest absolute deviation from the true capacity coefficients. The coefficients are displayed in lexicographic cardinality ordering: $\mu(\emptyset)$, $\mu(\{1\})$, $\mu(\{2\})$, $\mu(\{3\})$, $\mu(\{1,2\})$, $\mu(\{1,3\})$, $\mu(\{2,3\})$, $\mu(\{1,2,3\})$.

Key observations from the tables include:

- For all noise levels, PB-NN32's coefficient predictions are notably closer to the true capacity coefficients compared to those of PB-NN5.
- In the noiseless case, the k-interactive linear program precisely learns the value of true capacity coefficients.
- In general, the ChI-NN demonstrates superior performance in comparison to PB-NN5, although it does not quite attain the proficiency showcased by PB-NN32.
- In both the noisy and highly noisy cases, the coefficients predicted by PB-NN32 exhibit a remarkable alignment with those obtained from the k-interactive linear program. This observation holds special significance since, in this scenario, $k = n = 3$, resulting in the coefficients of the k-interactive linear program mirroring the optimal capacity. It's important to highlight that the introduced noise creates a difference between the true capacity used for generating the target variables and the capacity that yields the lowest loss in the training set, referred to as the optimal capacity.

Moreover, it's essential to highlight the persistent minor disparities observed in PB-NN's predictions, even for coefficients associated with the null set and universe set. These deviations arise due to the inherent limitations of enforcing equality constraints directly within deep neural networks. The method of multipliers, as outlined in section 3.3, only ensures that deviations from grounded and normalized constraints remain bounded by a predefined tolerance ϵ. Therefore, if achieving precise coefficient estimation holds paramount significance, it's prudent to consider alternative options, as PB-NN may not offer the most optimal solution for such requirements.

True	PB-NN5	PB-NN32	k-interactive	ChI-NN
0.0	0.0	0.0001	0.0	0.0
0.0807	0.0799	0.0809	0.0807	0.0801
0.1451	0.1349	0.1453	0.1451	0.175
0.3241	0.3141	0.3246	0.3241	0.3373
0.408	0.4094	0.4083	0.408	0.4659
0.4973	0.5004	0.4979	0.4973	0.5871
0.4362	0.449	0.4368	0.4362	0.5084
1.0	1.0001	1.0001	1.0	1.0

Table 4.1: Coefficients of capacity - noiseless case

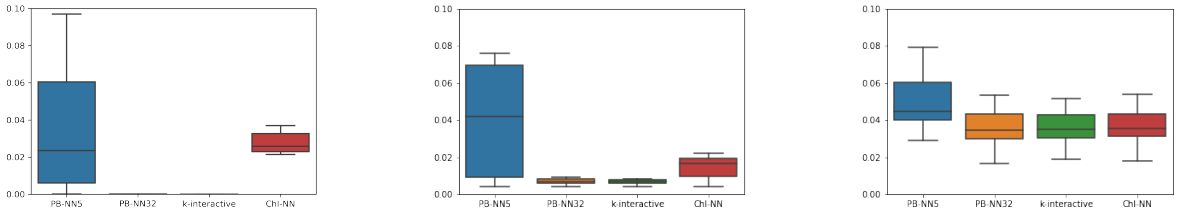
True	PB-NN5	PB-NN32	k-interactive	ChI-NN
0.0	-0.0	-0.0	0.0	0.0
0.2637	0.1421	0.253	0.2539	0.2685
0.0929	0.0713	0.084	0.084	0.0941
0.062	0.0	0.0364	0.0366	0.0444
0.4075	0.4875	0.4111	0.4104	0.4453
0.3653	0.6546	0.4295	0.4313	0.4289
0.1427	0.1507	0.1456	0.1454	0.1579
1.0	1.0	1.0001	1.0	1.0

Table 4.2: Coefficients of capacity - noisy case

To mitigate reliance on results from a single capacity instance, Figure 4.3 presents a box plot portraying the absolute deviations from true coefficients across the 10 randomly generated capacities. This visualization reinforces and extends insights gained from examining coefficient values in the tables.

True	PB-NN5	PB-NN32	k-interactive	ChI-NN
0.0	0.0	-0.0001	0.0	0.0
0.1677	0.1738	0.1717	0.1718	0.1786
0.2127	0.2697	0.1733	0.1727	0.2663
0.1971	0.1298	0.128	0.1268	0.1297
0.6923	0.6519	0.6569	0.6585	0.6447
0.2908	0.5099	0.5132	0.511	0.5039
0.66	0.7552	0.7963	0.7966	0.7939
1.0	1.0	1.0	1.0	1.0

Table 4.3: Coefficients of capacity - very noisy case



(a) noiseless case (b) noisy case (c) very noisy case

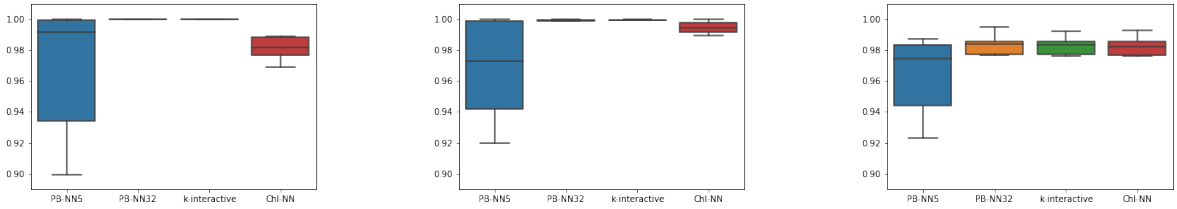
Figure 4.3: Deviation from true capacity coefficients, $n = 3$

Illustrated in Figure 4.4 and 4.5 are the coefficients of determination attained by the different algorithms in the test and training sets, respectively. Given the nature of this small universe, where 100 samples suffice to uniquely define the optimal capacity, no novel insights emerge concerning relative algorithmic performance. The algorithms that excel in accurately learning the optimal capacity also exhibit the highest coefficients of determination.

Nevertheless, several noteworthy observations can be made:

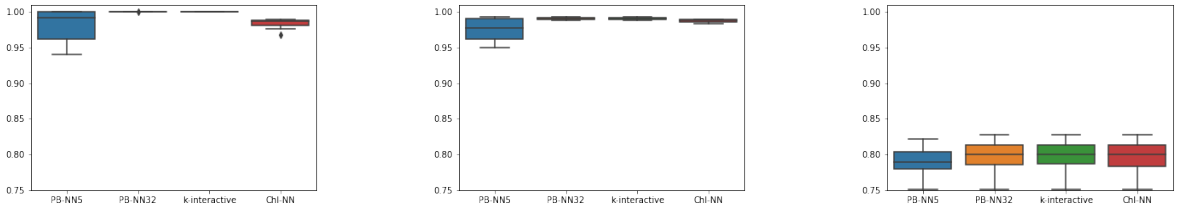
- Across all methods, the coefficient of determination in the training set experiences a pronounced decline in the very noisy case.
- In contrast, the R^2 in the test set demonstrates only a marginal decrease. This suggests that all algorithms manifest robustness in the presence of Gaussian noise.

In addition, a specific instance of the experiment revealed that PB-NN5 exhibited suboptimal performance, achieving an R^2 of only 47% on the training set and 31% on



(a) noiseless case (b) noisy case (c) very noisy case

Figure 4.4: Coefficient of Determination - Test set, $n = 3$



(a) noiseless case (b) noisy case (c) very noisy case

Figure 4.5: Coefficient of Determination - Training set, $n = 3$

the test set. This outcome, excluded from the preceding figures due to outlier removal, highlight a significant deviation from the targeted level of accuracy.

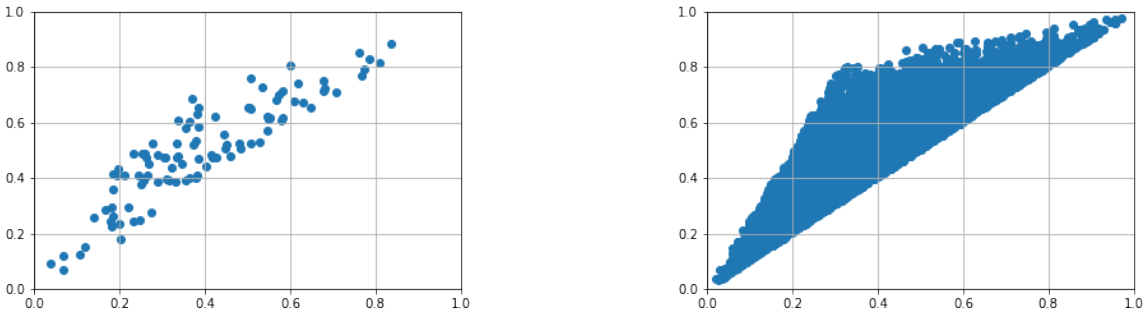
Table 4.4 further elucidates the issue by revealing the coefficients of the true capacity μ and the predicted capacity produced by PB-NN5. The predicted coefficients by PB-NN5 notably overestimate the true capacity coefficients. This discrepancy is reflected in Figure 4.6, where a clear bias towards higher predictions on both the training and test sets is evident.

To understand this phenomenon, we hypothesize that the algorithm may have converged to a suboptimal local minimum. In pursuit of confirming this hypothesis, we executed the algorithm 20 times from distinct random initializations. The results of this stability experiment are depicted in Figure 4.7. Remarkably, PB-NN5 achieved convergence to favorable local minima in 17 out of the 20 runs, displaying high R^2 values on both the training and test sets. Meanwhile, PB-NN5 converged to unfavorable local minima in the remaining 3 runs, providing empirical support for our conjecture. It is noteworthy that the occurrence of converging to a bad local minimum is an inherent limitation of our proposed approach. It is worth highlighting that such convergence challenges were not encountered with PB-NN32.

This insight underscores the potential drawback of local minima during optimization and highlights the need for careful initialization strategies to mitigate the risk of undesirable outcomes. The superior stability observed with PB-NN32 also suggests the importance of neural architecture in influencing convergence behavior and performance.

True	PB-NN5
0.0	0.0001
0.1809	0.4064
0.1335	0.171
0.2215	0.4227
0.5513	0.5773
0.3216	0.8291
0.2619	0.5936
1.0	1.0

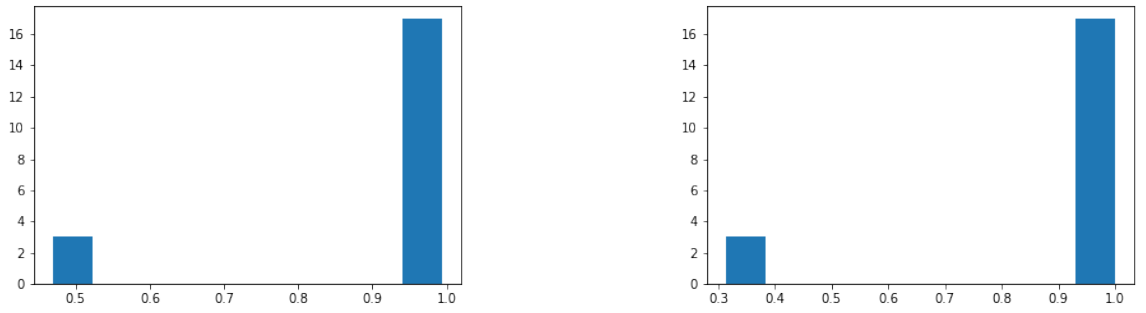
Table 4.4: Coefficient of the capacity in the case that PB-NN5 converged to a bad local minimum



(a) Training set

(b) Test set

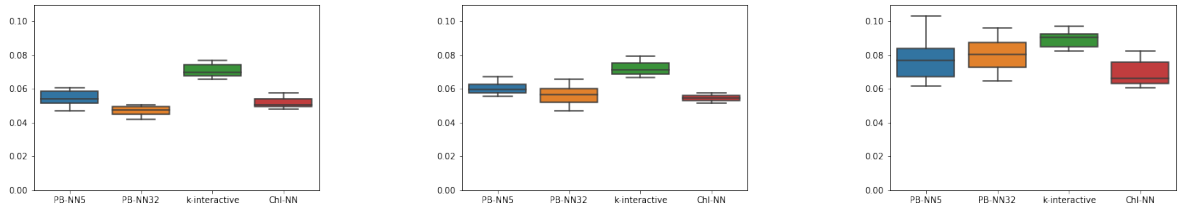
Figure 4.6: Plot of the true value versus the predicted value for PB-NN5 in the case it converged to a bad local minimum.



(a) Training set

(b) Test set

Figure 4.7: Histogram of the coefficient of determination across 20 random initializations



(a) noiseless case (b) noisy case (c) very noisy case

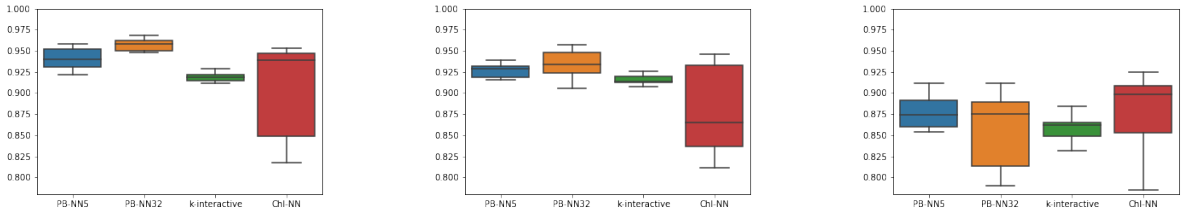
Figure 4.8: Deviation from true capacity coefficients, $n = 8$

4.2.2 8 Inputs

In the subsequent section, we present the outcomes of our experiment with a universe comprising 8 inputs. In this particular context, PB-NN5 is characterized by 81 parameters, PB-NN32 possesses 1377 parameters, and the capacity is defined by $2^8 = 256$ parameters.

Figure 4.8 illustrates a box plot depicting the absolute deviations from the true capacity. Noteworthy observations include:

- The k-interactive linear program consistently exhibits the largest deviation across all noise levels. This result aligns with expectations, as the randomly generated capacities are not 5-interactive. Consequently, the k-interactive linear program lacks the modelling power to effectively learn the underlying capacity structure.
- Across all methods, a substantially greater deviation from the true capacity coefficients is observed compared to the scenario presented in Figure 4.3, which pertains to the case of a universe with only 3 inputs. This outcome is anticipated, attributed to the challenge of training with just 100 samples in a universe of size 8. The potential non-uniqueness of the minimizer implies the existence of multiple capacities capable of explaining the training data.
- Generally, PB-NN5 demonstrates inferior performance in comparison to PB-NN32. However, it is notable that in the very noisy case, the two methods exhibit similar performance.
- Particularly noteworthy is the commendable performance achieved by ChI-NN in the very noisy scenario.



(a) noiseless case (b) noisy case (c) very noisy case

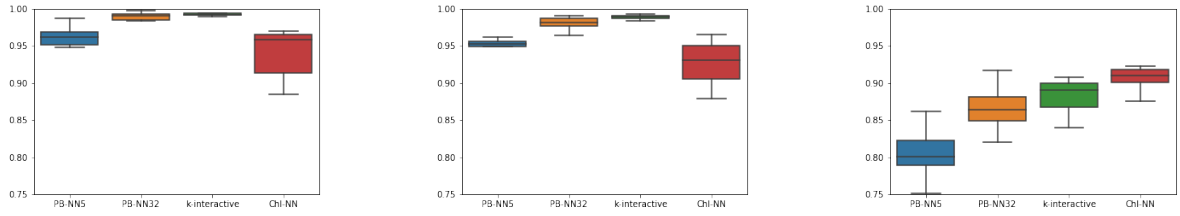
Figure 4.9: Coefficient of Determination - Test set, $n = 8$

Figure 4.9 and Figure 4.10 display the coefficients of determination obtained by all algorithms on the test and training sets, respectively. Several observations emerge from these figures:

- The k-interactive linear program demonstrates exceptional performance on the training set, establishing itself as the top-performing method in both noiseless and noisy scenarios. However, its performance is outpaced by PB-NN5 and PB-NN32 on the test set. We propose that this divergence can be attributed to the existence of 5-interactive capacities that adeptly capture the intricacies of the training data. Nonetheless, when extended to the test set, these capacities encounter challenges in generalization, particularly due to the fact that the actual capacity is not inherently 5-interactive.
- The ChI-NN method demonstrates noteworthy variability in the achieved R^2 values across the 10 random capacities, signifying a considerable variance in its performance.
- PB-NN32 consistently outperforms PB-NN5 on the training set, underscoring its superiority in this regard. However, their performance on the test set does not exhibit a significant difference.

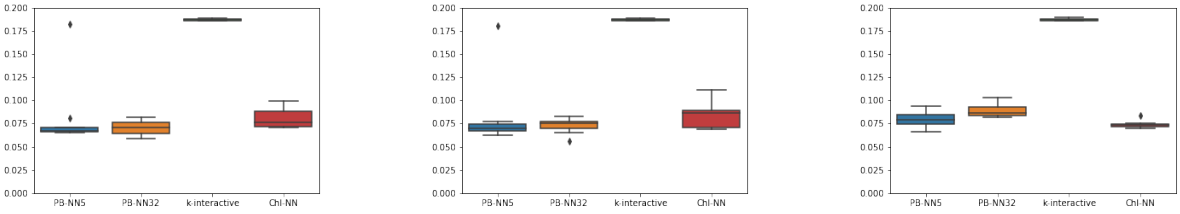
4.2.3 15 Inputs

In the forthcoming section, we elucidate the outcomes derived from an experiment undertaken in a universe characterized by 15 inputs. In this setting, PB-NN5 has 116 parameters and PB-NN32 has 1601 parameters, both significantly less than the number of parameters required to define the capacity $2^{15} = 32768$. Figure 4.11 portrays the absolute deviation



(a) noiseless case (b) noisy case (c) very noisy case

Figure 4.10: Coefficient of Determination - Training set, $n = 8$



(a) noiseless case (b) noisy case (c) very noisy case

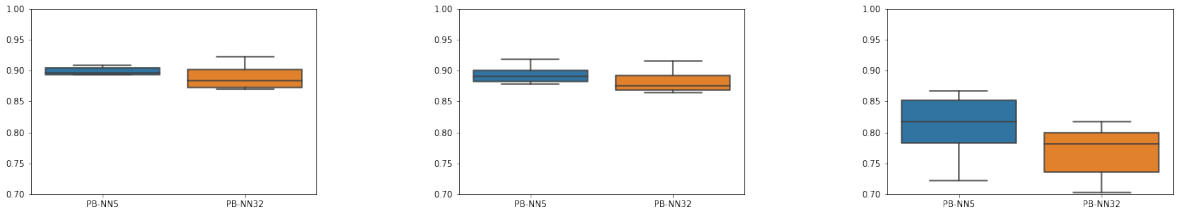
Figure 4.11: Deviation from true capacity coefficients, $n = 15$

of the capacity coefficients computed by the algorithms from the coefficients of the true capacity employed in generating the target labels.

Several noteworthy observations emerge from our analysis:

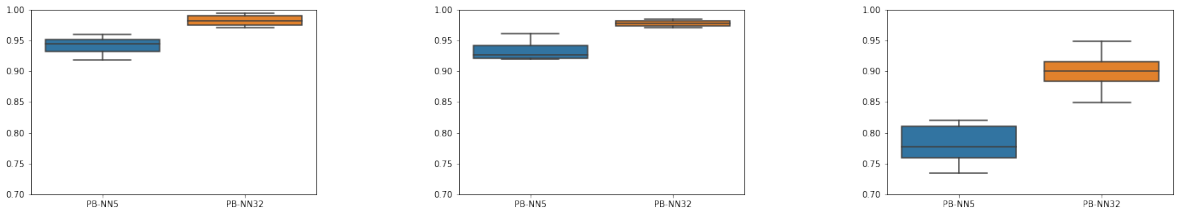
- The k-interactive linear program exhibits a notably higher deviation from the true coefficients compared to the other methods.
- In the noiseless and noisy scenario, two distinct extreme outliers are apparent for PB-NN5. These anomalies are consistent with the discussion on unfavorable local minima presented in Section 4.2.1.

Figures 4.12 and 4.5 offer a comprehensive illustration of the coefficient of determination (R^2) achieved by PB-NN5 and PB-NN32, showcased across both the training and test datasets. Remarkably, both neural architectures exhibit impressive performance, with notably high R^2. This achievement gains even greater significance when considering the constraint of having only 100 training samples, and the fact that both architectures possess



(a) noiseless case (b) noisy case (c) very noisy case

Figure 4.12: Coefficient of Determination - Test set, $n = 15$



(a) noiseless case (b) noisy case (c) very noisy case

Figure 4.13: Coefficient of Determination - Training set, $n = 15$

significantly fewer parameters compared to the demanding task of defining the capacity, which necessitates 32768 parameters.

PB-NN32 demonstrates superior performance on the training set when compared to PB-NN5, yet the situation shifts on the test set, where PB-NN5 emerges as the more adept performer. This discrepancy potentially suggests overfitting tendencies in PB-NN32. It is pertinent to note that in the noiseless and noisy case, PB-NN5's convergence to an unfavorable local minimum for a specific random capacity substantially impacts the average R^2. Consequently, PB-NN32 exhibits a higher average R^2 performance in noiseless and noisy scenarios, albeit with a lower median performance.

Figure 4.14 and Figure 4.15 portray the coefficient of determination (R^2) for the k-interactive linear program. It is particularly noteworthy that the k-interactive linear program yields a negative R^2 value on both the training and test sets. This implies that its performance is inferior to the constant model predicting the average target. We propose two potential explanations for this unsatisfactory outcome.

Firstly, the constraint enforcing a 5-additive measure proves to be overly restrictive, especially when the true capacity employed in generating the target labels adheres to a

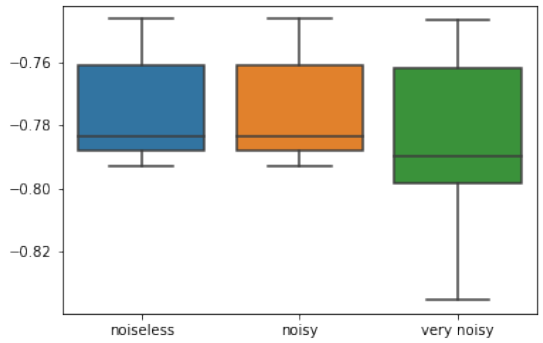


Figure 4.14: Coefficient of Determination - Test set - k-interactive

more general fuzzy measure. Consequently, the k-interactive linear program encounters limitations in capturing the underlying capacity structure effectively.

Secondly, given the context of having only 100 samples, the maximum number of data-supported variables stands at 1500, whereas the universe with 15 inputs entails a total of $2^{15} = 32768$ variables. This implies that a significant portion of these variables are classified as data-unsupported variables. For this subset of variables, the k-interactive linear program employs an imputation strategy whereby $\mu(A) = \max_{B \in \mathcal{P}, B \subset A} \mu(B)$, with $\mathcal{P}$ representing the set of data-supported variables, to align with k-maxitive measures [10]. We propose that, within the confines of this experiment, this imputation strategy proves to be suboptimal.

In contrast, we speculate that PB-NN employs a machine learning-based imputation approach, effectively learning the optimal imputation patterns from the data, akin to the concept proposed in [37]. This facet contributes to its enhanced performance, particularly on data-unsupported variables. Nonetheless, additional rigorous testing is warranted to substantiate this hypothesis.

A similar pattern of underperformance in this expansive universe can be observed with the CHI-NN, albeit to a lesser degree compared to the k-interactive linear program, as evidenced in Figures 4.16 and 4.17. The exact cause of this underperformance by the ChI-NN remains ambiguous. We speculate that the diminished performance could potentially stem from the substantial presence of data unsupported variables. However, a comprehensive investigation into the precise reasons behind the ChI-NN's underperformance falls beyond the scope of this particular research project.

It is important to emphasize that the primary objective of this experiment was to

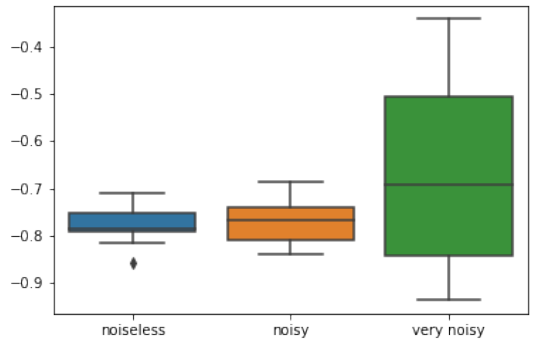


Figure 4.15: Coefficient of Determination - Training set - k-interactive

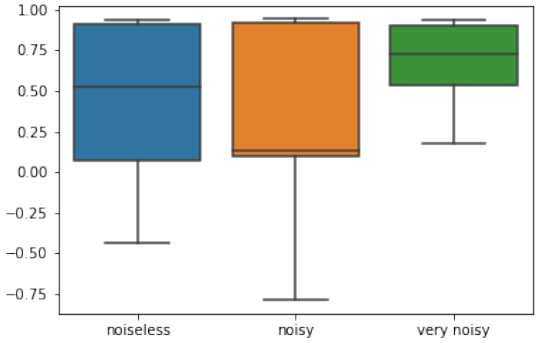


Figure 4.16: Coefficient of Determination - Test set - ChI-NN

demonstrate the ability of PB-NN in accurately learning capacities from data. We firmly believe that this goal has been successfully achieved.

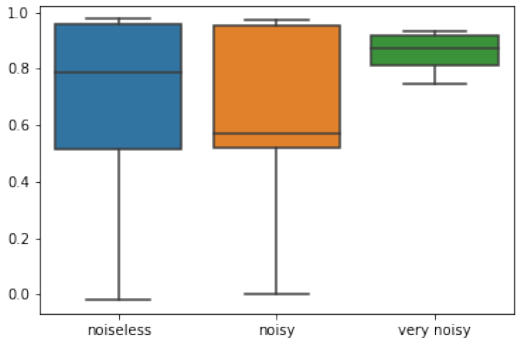


Figure 4.17: Coefficient of Determination - Training set - ChI-NN

Chapter 5

Conclusion

In this research project, we introduced a novel model class based on neural networks for approximating pseudo-Boolean functions. This departure from the prevalent polynomial-based models in the current literature, aimed to learn capacities from data, marks a significant advancement. Our approach enables the efficient learning of capacities from data within universes containing up to 100 variables, a substantial stride beyond the existing state-of-the-art, which is limited to 30 variables. Moreover, our method adeptly learns capacities across varying noise levels and universe sizes, exhibiting promising outcomes even in scenarios characterized by a substantial presence of data-unsupported variables.

It is important to note that while the proposed method stands as a powerful tool, it is not intended to replace classical techniques for learning capacities from data. These classical methods, having matured over three decades, often meet the robustness and computational efficiency standards requisite in practical applications. Our proposition advocates for the utilization of deep neural networks specifically in large universes scenarios where existing drastic simplifying assumptions about the capacities are unsuitable. Such scenarios frequently arise in diverse fields including pattern recognition, machine learning, signal/image processing, and computer vision.

Though we presented a series of promising results, it is evident that this research project has sparked more questions than it has answered. The potential range of pseudo-Boolean functions that deep neural networks can approximate when the number of parameters p in the neural network is significantly fewer than the parameters required to define the capacity 2^n remains a mystery. Additionally, the seemingly robust performance of neural networks in handling data-unsupported variables warrants deeper investigation. Determining the optimal architecture for these networks, as well as identifying the requisite amount of

data for learning capacities in large universes, present further avenues for exploration. Addressing concerns about convergence to unfavorable local minima, refining termination criteria for inner optimization, and investigating the impact of regularization on neural network parameters are among the many intriguing questions that arise.

Moreover, it is unclear whether neural networks are the optimal model class for pseudo-Boolean functions. While we have demonstrated that deep neural networks scale better within large universes than polynomials, there may be other model classes that outperform neural networks. We believe that monotone machine learning models offer a promising avenue for future research and refer interested readers to the following review on the topic [16].

Furthermore, our work raises the challenge of enabling explainable AI with Choquet integral-based models in expansive universes. While useful metrics for explainable AI were proposed in [50], their exponential dependency on the input size n presents computational challenges in large universes. Developing heuristics that scale polynomially with n to accurately approximate these metrics is a critical endeavor. Drawing inspiration from heuristics employed in other contexts, such as [46], could potentially offer valuable insights.

We also envisage the extension of a similar approach, proposed in this research, to address the optimal transport problem for capacities [28, 64]. In this context, the challenge involves finding a submodular capacity π that accepts as marginals given submodular capacities μ and ν, while minimizing the value of the Choquet integral evaluated at a cost function c. We believe that our approach could be adapted to solve this problem, albeit with necessary modifications to the loss function, constraint incorporation, and imposing concavity and monotonicy constraints on the neural network. However, addressing the open questions regarding the existence of submodular capacities that accepts as marginals submodular capacities and the interplay between submodularity and concavity remains pivotal before leveraging deep learning for this application.

In a broader perspective, we anticipate that leveraging deep neural networks for approximating pseudo-Boolean functions associated with optimal capacities holds substantial promise for impactful advancements across diverse fields where capacities and the related concept of fuzzy integrals find utility.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2016.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2623–2631, 2019.

[3] Brandon Amos, Lei Xu, and J. Zico Kolter. Input convex neural networks. *34th International Conference on Machine Learning, ICML 2017*, 1:192–206, 2017.

[4] Norman Archer and Shouhong Wang. Application of the Back Propagation Neural Network Algorithm with Monotonicity Constraints for Two-Group Classification Problems. *Decision Sciences*, 1993.

[5] Francis Bach. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends in Machine Learning*, 6(2-3):145–373, 2013.

[6] Gleb Beliakov. Construction of aggregation functions from data using linear programming. *Fuzzy Sets and Systems*, 160(1):65–75, 2009.

[7] Gleb Beliakov and Dmitriy Divakov. Aggregation with dependencies: Capacities and fuzzy integrals. *Fuzzy Sets and Systems*, 446:222–232, 2022.

[8] Gleb Beliakov and Simon James. Citation-based journal ranks: The use of fuzzy measures. *Fuzzy Sets and Systems*, 167(1):101–119, 2011.

[9] Gleb Beliakov, Simon James, and Jian Zhang Wu. *Discrete Fuzzy Measures, Computational Aspects*. 2020.

[10] Gleb Beliakov and Jian Zhang Wu. Learning fuzzy measures from data: Simplifications and optimisation strategies. *Information Sciences*, 494:100–113, 2019.

[11] Dimitri P. Bertsekas. Multiplier methods: A survey. *Automatica*, 12(2):133–145, 1976.

[12] Pratik Bhowal, Subhankar Sen, Juan D. Velasquez, and Ram Sarkar. Fuzzy ensemble of deep learning models using choquet fuzzy integral, coalition game and information theory for breast cancer histology classification. *Expert Systems with Applications*, 190(September 2021), 2022.

[13] Steven Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[14] Tomasa Calvo and Bernard de Baets. Aggregation Operators Defined by k-Order Additive/Maxitive Fuzzy Measures. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06, 1998.

[15] Kevin Canini, Andrew Cotter, Maya R. Gupta, Mahdi Milani Fard, and Jan Pfeifer. Fast and flexible monotonic functions with ensembles of lattices. *Advances in Neural Information Processing Systems*, 1(Nips):2927–2935, 2016.

[16] José Ramón Cano, Pedro Antonio Gutiérrez, Bartosz Krawczyk, Michał Woźniak, and Salvador García. Monotonic classification: An overview on algorithms, performance measures and data sets. *Neurocomputing*, 341:168–182, 2019.

[17] Gustave Choquet. Theory of capacities. *Annales de l'Institut Fourier*, 5:131–295, 1954.

[18] Djork Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pages 1–14, 2016.

[19] Elías F. Combarro, Julen Hurtado de Saracho, and Irene Díaz. Minimals Plus: An improved algorithm for the random generation of linear extensions of partially ordered sets. *Information Sciences*, 501:50–67, 2019.

[20] Miguel Couceiro, Jean Luc Marichal, and Tamás Waldhauser. Locally monotone Boolean and pseudo-Boolean functions. *Discrete Applied Mathematics*, 160(12):1651–1660, 2012.

[21] Hennie Daniels and Marina Velikova. Monotone and Partially Monotone Neural Networks. *IEEE Transactions on Neural Networks*, 21(6):906–917, 2010.

[22] Arka Daw, Anuj Karpatne, William D. Watkins, Jordan S. Read, and Vipin Kumar. Physics-Guided Neural Networks (PGNN): An Application in Lake Temperature Modeling. *Knowledge-Guided Machine Learning*, pages 353–372, 2022.

[23] Alp Dener, Marco Andres Miller, Randy Michael Churchill, Todd Munson, and Choong-Seock Chang. Training neural networks under physical constraints using a stochastic augmented Lagrangian approach. pages 1–21, 2020.

[24] Subhrajit Dey, Rajdeep Bhattacharya, Samir Malakar, Seyedali Mirjalili, and Ram Sarkar. Choquet fuzzy integral-based classifier ensemble technique for COVID-19 detection. *Computers in Biology and Medicine*, 135(May), 2021.

[25] Pablo A. Estévez and Yoichi Okabe. On the Computational Power of Max-Min Propagation Neural Networks. *Neural Processing Letters*, 19(1):11–23, 2004.

[26] Ali Fallah Tehrani, Weiwei Cheng, Krzysztof Dembczyński, and Eyke Hüllermeier. Learning monotone nonlinear models using the Choquet integral. *Machine Learning*, 89(1-2):183–211, 2012.

[27] A. V. Fiacco and G.P. McCormick. *Nonlinear Programming: Sequential Unconstrained minimization Techniques*. Wiley, 1968.

[28] Mario Ghossoub, David Saunders, and Kelvin Shuangjian Zhang. Bounds on Choquet Risk Measures in finite product spaces with ambiguous marginals. 2023.

[29] Michel Grabisch. A new algorithm for identifying fuzzy measures and its application to pattern recognition. In *Proceedings of 1995 IEEE International Conference on Fuzzy Systems.*, volume 1, pages 145–150, 1995.

[30] Michel Grabisch. The representation of importance and interaction of features by fuzzy measures. *Pattern Recognition Letters*, 17(6):567–575, 1996.

[31] Michel Grabisch. *Set Functions, Games and Capacities in Decision Making*. 2016.

[32] Michel Grabisch, Ivan Kojadinovic, and Patrick Meyer. A review of methods for capacity identification in Choquet integral based multi-attribute utility theory. Applications of the Kappalab R package. *European Journal of Operational Research*, 186(2):766–785, 2008.

[33] Michel Grabisch and Christophe Labreuche. A decade of application of the Choquet and Sugeno integrals in multi-criteria decision aid. *Annals of Operations Research*, 175(1):247–286, 2010.

[34] Michel Grabisch and Jean Marie Nicolas. Classification by fuzzy integral: Performance and tests. *Fuzzy Sets and Systems*, 65(2-3):255–271, 1994.

[35] Michel Grabish. Modelling data by the Choquet Integral. In Vicenç Torra, editor, *Information Fusion in Data Mining*, pages 135–149. Springer, 2002.

[36] Maya Gupta, Andrew Cotter, Jan Pfeifer, Konstantin Voevodski, Kevin Canini, Alexander Mangylov, Wojciech Moczydlowski, and Alexander Van Esbroeck. Monotonic calibrated interpolated look-up tables. *Journal of Machine Learning Research*, 17:1–47, 2016.

[37] Muhammad Aminul Islam, Derek T Anderson, Anthony J Pinar, and Timothy C Havens. Data-Driven Compression and Efficient Learning of the Choquet Integral. *IEEE Transactions on Fuzzy Systems*, 26(4):1908–1922, 2018.

[38] Muhammad Aminul Islam, Derek T. Anderson, Anthony J. Pinar, Timothy C. Havens, Grant Scott, and James M. Keller. Enabling Explainable Fusion in Deep Learning with Fuzzy Integral Neural Networks. *IEEE Transactions on Fuzzy Systems*, 28(7):1291–1300, 2020.

[39] Xiaowei Jia, Jared Willard, Anuj Karpatne, Jordan Read, Jacob Zwart, Michael Steinbach, and Vipin Kumar. Physics guided RNNs for modeling dynamical systems: A case study in simulating lake temperature profiles. *SIAM International Conference on Data Mining, SDM 2019*, pages 558–566, 2019.

[40] James M. Keller and Jeffrey Osborn. Training the fuzzy integral. *International Journal of Approximate Reasoning*, 15(1):1–24, 1996.

[41] J Kiefer and J. Wolfowitz. Stochastic Estimation of the Maximum of a Regression Function. *Ann. Math. Statist*, 23(3):462–466, 1952.

[42] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

[43] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *Advances in Neural Information Processing Systems*, 2017-Decem:972–981, 2017.

[44] Xingchao Liu, Xing Han, Na Zhang, and Qiang Liu. Certified monotonic neural networks. *Advances in Neural Information Processing Systems*, 2020-Decem(NeurIPS), 2020.

[45] David Luenberger. *Introduction to Linear and Non- linear Programming*. Addison-Wesley, Reading, MA, 1973.

[46] Scott Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In *31st Conference on Neural Information Processing Systems*, 2017.

[47] Andres Mendez-Vazquez and Paul Gader. Maximum A Posteriori EM MCE Logistic LASSO for learning fuzzy measures. In *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*, pages 2007–2013, 2008.

[48] M. A. Miller, R. M. Churchill, A. Dener, C. S. Chang, T. Munson, and R. Hager. Encoder-decoder neural network for solving the nonlinear Fokker-Planck-Landau collision operator in XGC. *Journal of Plasma Physics*, 87(2):1–13, 2021.

[49] Javier Murillo, Serge Guillaume, and Pilar Bulacio. k-maxitive fuzzy measures: A scalable approach to model interactions. *Fuzzy Sets and Systems*, 324:33–48, 2017.

[50] Bryce Murray, Muhammad Aminul Islam, Anthony J Pinar, Timothy C Havens, Derek T Anderson, and Grant Scott. Explainable AI for Understanding Decisions and Data-Driven Optimization of the Choquet Integral. In *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–8, 2018.

[51] Vinod Nair and Geoffrey Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *International Conference on Machine Learning*, pages 807–814, 2010.

[52] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf Xamla, Edward Yang, Zach Devito, Martin Raison Nabla,

Alykhan Tejani, Sasank Chilamkurthy, Qure Ai, Benoit Steiner, Fang Lu, Bai Junjie, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems. *NeurIPS*, (NeurIPS):8026–8037, 2019.

[53] T. D. Pham and H. Yan. Color image segmentation using fuzzy integral and mountain clustering. *Fuzzy Sets and Systems*, 107(2):121–130, 1999.

[54] Anthony J Pinar, Joseph Rice, Lequn Hu, Derek T Anderson, and Timothy C Havens. Efficient Multiple Kernel Classification Using Feature and Decision Level Fusion. *IEEE Transactions on Fuzzy Systems*, 25(6):1403–1416, 2017.

[55] Stanton R. Price, Steven R. Price, and Derek T. Anderson. Introducing Fuzzy Layers for Deep Learning. *IEEE International Conference on Fuzzy Systems*, 2019-June, 2019.

[56] H Robbins and S Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[57] Davor Runje and Sharath M. Shankaranarayana. Constrained Monotonic Neural Networks. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[58] Sara Sangalli, Ertunc Erdil, Andreas Hoetker, Olivio Donati, and Ender Konukoglu. Constrained Optimization to Train Neural Networks on Critical and Under-Represented Classes. *Advances in Neural Information Processing Systems*, 30(NeurIPS):25400–25411, 2021.

[59] Grant J Scott, Kyle C Hagan, Richard A Marcum, James Alex Hurt, Derek T Anderson, and Curt H Davis. Enhanced Fusion of Deep Neural Networks for Classification of Benchmark High-Resolution Image Data Sets. *IEEE Geoscience and Remote Sensing Letters*, 15(9):1451–1455, 2018.

[60] Joseph Sill. Monotonic Networks. *Advances in neural information processing systems*, 10, 1997.

[61] Joseph Sill and Yaser Abu-Mostafa. Monotonicity Hints. In M C Mozer, M Jordan, and T Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1996.

[62] Tomohiro Takagi and Michio Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(1):116–132, 1985.

[63] Ali Fallah Tehrani, Christophe Labreuche, and H Eyke. Choquistic Utilitaristic Regression. *DA2PL-14*, (November), 2014.

[64] Vicenç Torra. The transport problem for non-additive measures. *European Journal of Operational Research*, (xxxx), 2023.

[65] Mikel Uriz, Daniel Paternain, Humberto Bustince, and Mikel Galar. A supervised fuzzy measure learning algorithm for combining classifiers. *Information Sciences*, 622:490–511, 2023.

[66] Mikel Uriz, Daniel Paternain, Iris Dominguez-Catena, Humberto Bustince, and Mikel Galar. Unsupervised Fuzzy Measure Learning for Classifier Ensembles From Coalitions Performance. *IEEE Access*, 8:52288–52305, 2020.

[67] J Wang and Z Wang. Detecting constructions of nonlinear integral systems from input-output data: an application of neural networks. In *Proceedings of North American Fuzzy Information Processing*, pages 559–563, 1996.

[68] W Wang. Genetic algorithms for determining fuzzy measures from data. *Journal of Intelligent & Fuzzy Systems*, 6(2):171–183, 1998.

[69] Xi Zhao Wang, Yu Lin He, Ling Cai Dong, and Huan Yu Zhao. Particle swarm optimization for determining fuzzy measures from data. *Information Sciences*, 181(19):4230–4252, 2011.

[70] Zhenyuan Wang, Kwong Sak Leung, and George J. Klir. Applying fuzzy measures and nonlinear integrals in data mining. *Fuzzy Sets and Systems*, 156(3):371–380, 2005.

[71] Seungil You, David Ding, Kevin Canini, Jan Pfeifer, and Maya R. Gupta. Deep lattice networks and partial monotonic functions. *Advances in Neural Information Processing Systems*, 2017-Decem(Nips):2982–2990, 2017.