Universitat Autònoma de Barcelona

Faculty of Science
Department of Mathematics

Bachelor's Thesis

Fuzzy Logic in Artificial Intelligence:

a study of fuzzy set theory and its applications to Explainable AI

Loredana Sandu

July, 2023

Supervised by
Dr. Pilar Dellunde

Co-supervised by
Dr. Wolfgang Pitsch

Abstract

This thesis explores the integration of fuzzy set theory and fuzzy logic into artificial intelligence (AI) systems, with an emphasis on enhancing their interpretability. We present the mathematical foundations of fuzzy set theory, and demonstrate its application to the design and implementation of a fuzzy rule-based system for a prediction task. We provide a mathematical description of *fuzzy neural networks*, an extension of classical neural networks that incorporates fuzzy logic operators. We apply both a classical and a fuzzy neural network to a prediction task, in order to compare their performance and investigate the advantages and limitations of the latter model. The fuzzy neural network shows similar performance to the classical neural network, while showing increased interpretability through the optimization of fuzzy logic rules. This reveals the potential of fuzzy logic to improve the interpretability of AI systems without significantly compromising their accuracy.

The thesis thereby investigates alternative approaches to the development of AI models, emphasizing the need for models that are both explainable and maintain high accuracy, a fundamental goal of the field of Explainable AI (XAI). Further implications of this research and suggestions for future work are also discussed. The thesis contributes to the growing discourse on the role of explainability in AI, setting a precedent for future research in the field.

Keywords — fuzzy logic, artificial intelligence, Explainable AI, fuzzy neural networks.

Acknowledgements

I wish to thank Pilar for her dedicated guidance through this journey of exploration, and for letting my curiosity flourish. Thank you for all your close attention and your broad openness. For encouraging the pursuit of my own ideas, and for helping me find my way through the complexities. Thank you for your unceasing encouragement, for sharing your perspectives with me and for all the meaningful advice, which has proven very valuable for this work and beyond.

I would also like to thank Vicent Costa for his insightful feedback, and for always listening to what I had to say. For prompting new ways of thinking, both in the big picture and in the details. The insightful suggestions of both Vicent and Pilar have uncovered technicalities and conceptual perspectives that have been very valuable for this work, and will continue to be so in the future.

I also wish to thank Wolfgang for always directing my attention towards the meaningful details, while also keeping a broad perspective. Thank you for your encouragement and for your valuable feedback, which has helped me grow both technically and personally, during this project and throughout the course of all the degree.

I would like to acknowledge all the people who came before them and have contributed throughout the years, enhancing my curiosity and interest for the fascinating realm of Mathematics long before the degree. A special thanks to Maria, Mercè, Xesco, Marina, Joan Carles and Àngels for their stimulating instruction and encouragement throughout the years. They laid the foundations that led me to pursue this path.

Thanks to my friends for listening to all my ramblings about these topics with all their interest and enthusiasm. For standing by me, and always spurring me towards my goals, even when the code is full of bugs.

Above all, my deepest gratitude goes to my parents. For their unparalleled fortitude and persistence, for their wise and invaluable guidance, and for their unwavering support. For sparking my curiosity beyond measure from very early ages, and for ensuring the opportunity for me to do this, unconditionally. *Hai hadem.*

Contents

List of Figures

List of Tables

1 Introduction

Fuzzy logic is an extension of classical logic that accommodates degrees of truth. Extending the binary integer values 0 and 1 of classical (or Boolean) logic, fuzzy logic allows for the representation of truth values in the range $[0, 1] \subset \mathbb{R}$, where 1 represents "totally true", 0 represents "totally false", and $(0, 1)$ represents intermediate degrees of truth. It is based on the concept of *fuzzy set*, first introduced by L. A. Zadeh in 1965 [1], who proposed fuzzy set theory as a mathematical way of handling uncertainty and imprecision inherent in real-world data and decision-making processes. For instance, fuzzy logic can model vague concepts such as "tall", "young" or "rich", which are often difficult to define without ambiguity.

Fuzzy logic is inspired by human reasoning, which is often based on partial truths and degrees of certainty. It is therefore well-suited for processing the complex and vague information often encountered in real-world applications, such as medical diagnosis [2–4], image processing [5–7], natural language processing [8, 9], or forecasting of natural phenomena [10–13].

Fuzzy logic has been applied to a wide range of Artificial Intelligence (AI) systems. We now present a brief historical background, covering the main applications of fuzzy logic in AI in the past decades. We discuss some of the current challenges and limitations in the field of AI and how fuzzy logic can contribute to addressing them, which will motivate the objectives and scope for the present work.

1.1 Applications of fuzzy logic in Artificial Intelligence: historical background and current challenges

The history of fuzzy logic and its applications is closely intertwined with the history of AI, with fuzzy logic techniques being used in the development of AI systems to address challenges related to uncertainty, imprecision and vagueness in data. Here we provide a brief overview of the main applications of fuzzy logic in AI in the past few decades. We outline the parallel development of both disciplines and explore their relationship[1].

The early stages of AI as a field, dating back to the 1950s and 1960s, were characterized by the development of rule-based AI systems which relied on the use of formal logic and symbolic manipulation [17–19]. It was during this period that fuzzy sets were introduced by L. A. Zadeh in 1965 [1], laying the foundations of fuzzy set theory. During this stage, the applications of fuzzy set theory in AI were limited, primarily due to the fact that this paradigm was still in its infancy. [20] was one of the first works that discussed the application of fuzzy logic in decision-making, which is a crucial aspect of AI systems.

During the 1970s and 1980s, AI was characterized by the development of knowledge-based AI systems and expert systems, which relied on the representation of domain-specific knowledge to make decisions [21, 22]. During this period, the application of fuzzy logic in AI was developed extensively, with fuzzy logic being used to represent and manipulate uncertain or imprecise knowledge [23, 24]. Fuzzy logic was applied in the development of control systems and expert systems in various domains, including medical diagnosis [25, 26], emergency management in fields such as nuclear technology, aviation or medicine [27], autonomous navigation [28], and control of robots and other industrial processes [29].

During the second half of the 1980s and in the 1990s, the development of AI systems was characterized by a rise in the use of neural networks[2]. These systems were inspired by the human brain and relied on the use of statistical techniques to learn from data [30]. At this stage, fuzzy logic found new applications, and was integrated with neural networks to give rise to fuzzy neural networks [31, 32].

[1] A more detailed overview of the history and development of AI can be found in [14, pp. 17-27]. [14] also provides an in-depth coverage of the main trends and practices in the field of AI throughout its history. For a comprehensive account of the development of AI from its origins to the early 1990s, consider [15]. For a historical account of fuzzy logic, its mathematical treatment and its applications, the reader is referred to [16].

[2] An introduction to neural networks can be found in section 3.1 of this text.

These hybrid systems combined the advantages of both paradigms: the flexibility and adaptability of neural networks (with their ability to learn from data), and the ability of fuzzy logic to handle uncertainty and imprecision in data, to encode prior knowledge, and to define interpretable if-then rules in the inference process [33]. As neural networks began to be used successfully in different applications in the 1990s, fuzzy neural networks were also applied in various domains. Some of these applications include the design of control systems such as for vehicle or robot control [34, 35], and a variety of regression and classification problems such as image classification and turbidity optimization [33, 36, 37].

In the last decades, from the 2000s to the present, probabilistic approaches have had a major role in the design of AI systems, in contrast to the previous logic-based approaches [38, 39]. In the last decade in particular, the use of neural networks and deep learning techniques has increasingly dominated AI [38, 40]. These techniques have led to significant progress in various fields, including natural language processing [41, 42], speech recognition [43], visual object recognition [44, 45], robotics and autonomous systems [46, 47], drug discovery [48], and genomics and computational biology [49].

However, modern techniques in deep learning also pose certain challenges and limitations. We now discuss some of the key current challenges, and how fuzzy logic can contribute to addressing them. We include references that demonstrate the integration of fuzzy logic with AI systems to approach these challenges, and the recent advances in the combination of fuzzy logic with deep learning techniques.

Current challenges and limitations of AI and deep learning techniques

First, deep learning models typically require large amounts of data to be trained, and can be sensitive to small changes in the input data [40]. In many real-world applications, however, data is often scarce, it can be noisy or incomplete, or it can change dynamically to reflect new information [38, 50]. Fuzzy logic can be used to address these challenges by incorporating prior knowledge into the training process, and by using fuzzy sets to represent imprecise or vague data [32, 51].

Second, deep learning models often lack interpretability and transparency due to their complex architectures and lack of human-understandable inference processes. This lack of interpretability can be problematic in many domains such as medical diagnosis, autonomous driving, robotic assistance, or financial decision-making, which are regulated and where safety plays an important role [52]. Fuzzy logic can address this limitation by offering transparent and human-understandable inference processes through the use of predefined fuzzy logic rules, or through the automatic generation of new fuzzy logic rules [32, 53, 54]. This approach offers an increased interpretability of the decisions made by the AI system, and has already been applied to fields such as personalised medicine and genetics [55].

Third, many of the most successful deep learning models are based on mathematical and computational methods which may not always align with human reasoning. For instance, the use of activation functions to introduce non-linearity in neural networks may not directly replicate the mechanisms of human thought processes [56, 57]. However, for reasons similar to the ones discussed above, sometimes it may be desirable to incorporate techniques into AI systems that align more closely with human reasoning. Fuzzy logic can be used to address this limitation by using fuzzy logic rules to incorporate human reasoning processes into the design of AI systems, or generate these rules and processes automatically [32, 53, 54, 58].

Last, it is worth noting that fuzzy logic can be integrated into a wide range of modern AI techniques and architectures. Many combinations have been explored in the literature[3], resulting in significant improvements in different aspects of AI systems, like accuracy, robustness, and interpretability. For instance, fuzzy logic has been integrated with convolutional neural networks (CNNs) [60, 61], recurrent neural networks (RNNs) [62, 63], Autoencoders (AEs) [64, 65], generative adversarial networks (GANs) [66], reinforcement learning (RL) models [67–69], and genetic algorithms [70, 71].

[3]An in-depth review of hybrid models that integrate fuzzy systems with modern AI techniques, covering a wide range of procedures and applications, can be found in [59].

1.2 Explainable AI: background and motivation

As seen above, the lack of interpretability, transparency and alignment with human reasoning is one of the main limitations of modern AI systems. This has become the main focus of a new field of research known as *Explainable AI* (XAI).

The concept of explainability of AI systems involves the following main aspects [52, 72]:

(i) the ability to explain the decisions made by the system and the underlying phenomena which led to these decisions, in a way that is understandable to humans (transparency),

(ii) preventing the model from perpetuating any bias present in the data or arising from the training process (bias),

(iii) ensuring that the model takes fair decisions in the context of the application (fairness),

(iv) ensuring that the model is safe and reliable (safety).

In essence, all these aspects are related to the *interpretability* of the model, which is the extent to which the model and the predictions made by the model are human-understandable [52]. One of the most addressed discussions in the literature is in relation to how the interpretability of a model can be measured [73]. In this work, we regard the interpretability of AI models as the extent to which the operations and predictions they perform can be explained in terms of human-understandable rules.

The interpretability of AI systems often comes with a decrease in their accuracy in terms of predictive power. Therefore, one of the aims of XAI is to develop AI systems that are both explainable and maintain a high accuracy, comparable to that of previous less interpretable models[4] [52]. In addition, it has been criticized in the literature that even supposedly interpretable models can become difficult to interpret when they are high-dimensional [75]. These considerations will be taken into account in this work, by comparing the accuracy of AI models which integrate fuzzy logic to the accuracy of previous less interpretable models, and by discussing how the dimensionality of the models can affect both their performance and their interpretability.

1.3 Objectives, scope and structure of this study

The main goal of this work is to explore the incorporation of fuzzy set theory and fuzzy logic into AI systems, and to investigate the potential benefits of such combination. In particular, we will focus mainly on the use of fuzzy logic operators to develop interpretable and explainable AI systems.

To this end, our first objective is to provide an introduction to the mathematical foundations of fuzzy set theory, with a particular focus on fuzzy set operations, which model the *and*, *or* and *not* logical operators. We also seek to investigate the early uses of fuzzy set theory in AI, namely in the development of rule-based systems, illustrating how fuzzy logic is used to encode prior knowledge and to define interpretable if-then rules in the inference process. We demonstrate the design and implementation of a fuzzy rule-based system for the prediction of crop yields. Section 2 is dedicated to these objectives.

Our next objective is to explore the incorporation of fuzzy logic in one of the primary models in modern AI: neural networks. We seek to investigate the use of fuzzy logic operators in the design of interpretable neural networks. We will see how the resulting model, known as *fuzzy neural network*, combines the flexibility and adaptability of classical neural networks (with their ability to learn from data), with the improved interpretability and human-like reasoning which fuzzy logic can provide.

We demonstrate the design and implementation of both a classical and a fuzzy neural network for the prediction of crop yields, and compare the different systems implemented for the same task. Through this illustrative application, we will discuss different advantages and limitations of fuzzy neural networks. The mathematical foundations and the implementation of these models will be presented in section 3, and further discussion will be provided in section 4.

[4] For a survey of the main methods and techniques used in XAI, the reader is referred to [52, 74].

2 Fuzzy set theory

The aim of this section is to provide an introduction to fuzzy set theory. We provide a formal description of the main objects that fuzzy set theory addresses, *fuzzy sets*, and a survey of the main fuzzy set operations.

We also present an early use case of fuzzy set theory in AI, namely in the development of fuzzy rule-based systems. We demonstrate the design and implementation of a fuzzy rule-based system for the prediction of crop yields.

We start by reviewing notions of classical set theory that will later be extended to fuzzy set theory.

2.1 From crisp sets to fuzzy sets: an overview

It is standard in the literature to refer to classical non-fuzzy sets as *crisp sets*, an expression first introduced by L. A. Zadeh in 1965 [76]. The elements of a crisp set are referred to as *crisp elements*.

The notion of *characteristic function* will be particularly useful in our text:

Definition 1. Let X be a universal set[5], and consider a crisp set $A \subseteq X$. The *characteristic function* of A is the function $\chi_A : X \to \{0, 1\}$ such that, for all $x \in X$,

$$\chi_A(x) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{if } x \notin A. \end{cases}$$

This function declares which items in X are elements of the set A, assigning *crisp* boolean values $\{0, 1\}$ to each item in X. There is therefore a one-to-one correspondence between crisp sets and their characteristic functions:

Proposition 2. Let X be a universal set. For any two crisp sets $A, B \subseteq X$ with characteristic functions χ_A and χ_B, respectively, we have

$$A = B \iff \chi_A = \chi_B.$$

The concept of characteristic function will be extended later to the notion of *membership function*, which assigns a *fuzzy* value in the interval of real numbers $[0, 1]$ to each element in X, that corresponds to its degree of membership to a *fuzzy set* considered.

Let us now review the main operations performed on crisp sets.

Definition 3. Let X be a universal set, and $A, B \subseteq X$ two crisp sets with characteristic functions χ_A and χ_B, respectively. The *intersection* of A and B is defined as the set of items in X that are items of both A and B:

$$A \cap B = \{\, x \in X \mid \chi_A(x) = 1 \text{ and } \chi_B(x) = 1 \,\}.$$

The *union* of A and B is defined as the set of items in X that are items of either A or B:

$$A \cup B = \{\, x \in X \mid \chi_A(x) = 1 \text{ or } \chi_B(x) = 1 \,\}.$$

And the *complement* of A is defined as the set of elements in X that are not elements of A:

$$A^c = \{\, x \in X \mid \chi_A(x) = 0 \,\}.$$

Remark 4. The characteristic functions of the intersection, union and complement of crisp sets, denoted as $\chi_{A \cap B}$, $\chi_{A \cup B}$ and χ_{A^c}, respectively, satisfy the following equalities for all $x \in X$:

[5] All universal sets considered in this work are crisp sets.

(i) $\chi_{A\cap B}(x) = \chi_A(x) \cdot \chi_B(x)$,

(ii) $\chi_{A\cup B}(x) = \chi_A(x) + \chi_B(x) - \chi_A(x) \cdot \chi_B(x)$,

(iii) $\chi_{A^c}(x) = 1 - \chi_A(x)$. △

Proposition 5. Let X be a universal set, and $A, B \subseteq X$ two crisp sets. Crisp set operations satisfy the following equalities, known as *De Morgan's laws*[6]:

$$(A \cap B)^c = A^c \cup B^c,$$
$$(A \cup B)^c = A^c \cap B^c.$$

Remark 6. Considering X the universal set, and $A, B \subseteq X$ two crisp sets, De Morgan's laws are equivalent to the following equalities, in terms of the characteristic functions of the sets:

$$\chi_{A\cap B}(x) = 1 - \chi_{A^c \cup B^c}(x), \tag{1}$$
$$\chi_{A\cup B}(x) = 1 - \chi_{A^c \cap B^c}(x), \tag{2}$$

for all $x \in X$. △

There is a natural connection between classical sets and its operations, and classical logic statements and operators [78, pp. 6, 7]. Given a universal set X and a logical statement $P(x)$ defined for all $x \in X$, $P(x)$ determines a crisp set $A \subseteq X$ that contains the elements $a \in X$ such that $P(a)$ is true, i.e. $A = \{a \in X \mid P(a)\}$. For example, if $X = \mathbb{Z}$ and $P(x)$ is the statement "x is an even number", then A is the set of all even numbers.

Then, for any $x \in X$, the truth value of $P(x)$ is determined by the value of the characteristic function of A at x, i.e. for any $x \in X$, $P(x)$ is true if and only if $\chi_A(x) = 1$. Furthermore, the operations of intersection, union and complement of crisp sets correspond to the logical operators of conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$), respectively, in the following way: given two logical propositions $P(x)$ and $Q(x)$ defined for every $x \in X$, and the corresponding crisp sets $A = \{a \in X \mid P(a)\}$ and $B = \{b \in X \mid Q(b)\}$, we have

$$A \cap B = \{c \in X \mid P(c) \wedge Q(c)\},$$
$$A \cup B = \{c \in X \mid P(c) \vee Q(c)\},$$
$$A^c = \{c \in X \mid \neg P(c)\}.$$

The operations of intersection, union and complement of crisp sets will be extended to fuzzy sets through operators known as *t-norm*, *t-conorm* and *fuzzy complement*. It will be seen that these operators correspond to membership functions themselves, in the same way that crisp set operations can be expressed in terms of characteristic functions. They assign a fuzzy value in $[0, 1]$ to each element in X, corresponding to the degree of membership of the element to the intersection, the union and the complement of fuzzy sets, respectively. We will be interested in pairs of t-norms and t-conorms that satisfy a generalization of De Morgan's laws which will be presented later in the text. The relation between these operators and logical operators will also be presented, with t-norms, t-conorms and fuzzy complements corresponding to $\wedge$, $\vee$ and $\neg$, respectively.

2.2 Fuzzy sets

The main objects of study of fuzzy set theory are *fuzzy sets*, first introduced in [1].

Definition 7. Given a crisp set X, a *fuzzy set* A in X is a set of ordered pairs

$$A = \{(x, \mu_A(x)) \mid x \in X\},$$

where $\mu_A : X \to [0, 1] \subset \mathbb{R}$ is called *membership function*.

[6] For a proof of De Morgan's laws in the context of set theory, see [77, pp. 6-7].

The membership function assigns values to the elements of the universal set X that correspond to their degree of membership to the set A considered.

Remark 8. If we have $\mu_A(x) \in \{0, 1\}$ for all $x \in X$, then the fuzzy set A is, in particular, a crisp set. In this case, the membership function μ_A is the characteristic function of A (see Definition 1).

Therefore, fuzzy sets are a generalization of crisp sets, and the membership function of a fuzzy set is a generalization of the characteristic function of a crisp set. △

Example 9 [79]**.** We define a fuzzy set that represents the medical concept of "high fever", based on a person's temperature. According to [79], from the medical point of view, a person with a temperature higher than 39.0 °C has a high fever with degree of certainty 1 (that is, they surely have a high fever). A person whose temperature (in °C) is in the interval $[38.5, 39.0]$ has a high fever with some degree of certainty in $[0, 1]$. And a person with a temperature lower than 38.5 C° has a high fever with degree of certainty 0 (that is, they surely do not have a high fever).

Consider the universal set $X \subset \mathbb{R}$ to be the set of all possible temperatures that a person can have, in degrees Celsius (°C). In this example, we will consider $X = [34.0, 42.0]$. We define the fuzzy set HF, that represents "high fever", as $HF = \{(x, \mu_{HF}(x)) \mid x \in X\}$, where $\mu_{HF} : X \to [0, 1]$ is the membership function of HF, defined as follows:

$$\mu_{HF}(x) = \begin{cases} 0, & \text{if } x < 38.5, \\ \frac{x-38.5}{0.5}, & \text{if } 38.5 \leq x \leq 39.0, \\ 1, & \text{if } x > 39.0. \end{cases}$$

See Figure 1 for a graphical representation of μ_{HF}. △

Example 10 [80, Example 3.1]**.** Let us now define a fuzzy set representing the development of a country in terms of health, based on the life expectancy of its population. One approach to do so is to define an upper goalpost and a lower goalpost for the life expectancy of the population, and define the membership function of the fuzzy set in terms of these goalposts. With this approach, a country with a life expectancy above the upper goalpost can be considered highly developed in terms of health, and a country with a life expectancy below the lower goalpost can be considered highly underdeveloped in terms of health.

Consider the universal set $X \subset \mathbb{R}$ to be the set of all possible life expectancies that a country can have, in years. In this example, we will consider $X = [0.0, 100.0]$.

We define the fuzzy set H, that represents "developed countries in terms of health", as $H = \{(x, \mu_H(x)) \mid x \in X\}$, where $\mu_H : X \to [0, 1]$ is defined as

$$\mu_H(x) = \frac{1}{1 + e^{-a(x-b)}},$$

a logistic function with parameters a the slope of the function, and b the life expectancy corresponding to the midpoint of the function (i.e. such that $\mu_H(b) = 0.5$).

If we consider the upper goalpost to be 85 years, and the lower goalpost to be 25 years, then setting $a = 0.15$ and $b = 55$ yields the membership function

$$\mu_H(x) = \frac{1}{1 + e^{-0.15(x-55)}},$$

represented in Figure 2. △

Definition 11 [81, Def. 2-2]**.** Let A be a fuzzy set in a crisp set X, with membership function μ_A. The *support* of A, denoted $S(A)$, is the crisp set of elements that belong to A to some non-zero degree:

$$S(A) = \{x \in X \mid \mu_A(x) > 0\}.$$

Definition 12 [81, Def. 2-3]**.** Let A be a fuzzy set in a crisp set X, with membership function μ_A. Given $\alpha \in [0,1]$, the *α-level set* (or the *α-cut*) of A, denoted A_α, is the crisp set of elements that belong to A at least to the degree α:

$$A_\alpha = \{x \in X \mid \mu_A(x) \geq \alpha\}.$$

Also, the set $A'_\alpha = \{x \in X \mid \mu_A(x) > \alpha\}$ is called the *strong α-level set* or *strong α-cut* of the fuzzy set A.

Remark 13. Let A be a fuzzy set in a crisp set X. We observe that (i) $A'_0 = S(A)$, (ii) $A_0 = X$, and (iii) $A'_1 = \varnothing$. △

Example 14. Consider the fuzzy set HF that represents the medical concept of "high fever", defined in Example 9. The support of HF is

$$S(HF) = \{x \in X \mid x > 38.5\} = (38.5,\, 42.0]\,.$$

The α-level sets of HF are the following:

$$\begin{aligned} HF_0 &= X = [34.0,\, 42.0]\,, \\ HF_\alpha &= [0.5 \cdot \alpha + 38.5,\, 42.0] \quad \text{for } \alpha \in (0,1]. \end{aligned}$$ △

Example 15. Consider the fuzzy set H that represents "developed countries in terms of health", defined in Example 10, with membership function $\mu_H(x) = \dfrac{1}{1+e^{-0.15(x-55)}}$.

For $\alpha \in (0,1)$, the strong α-level set of H is

$$H'_\alpha = \left\{ x \in X \;\middle|\; x > 55.0 + \frac{1}{0.15} \cdot \ln\left(\frac{\alpha}{1-\alpha}\right) \right\} = \left(55.0 + \frac{1}{0.15} \cdot \ln\left(\frac{\alpha}{1-\alpha}\right),\, 100.0\right]. \qquad \triangle$$

2.3 Operations on fuzzy sets

Many possible operators have been proposed to extend the operations on classical sets to operations on fuzzy sets. To describe these specific operators, we first need to introduce the more general notion of t-norm and t-conorm.

Definition 16 [51, p. 62]**.** A *t-norm* is a function $t : [0,1] \times [0,1] \to [0,1]$ that satisfies the following properties:

(i) t is commutative: $t(x,y) = t(y,x)$ for all $x, y \in [0,1]$.

(ii) t is associative: $t(x, t(y,z)) = t(t(x,y), z)$ for all $x, y, z \in [0,1]$.

(iii) 1 is the identity element: $t(x,1) = t(1,x) = x$ for all $x \in [0,1]$.

(iv) t is monotonous: if $y \leq z$, then $t(x,y) \leq t(x,z)$ for all $x, y, z \in [0,1]$.

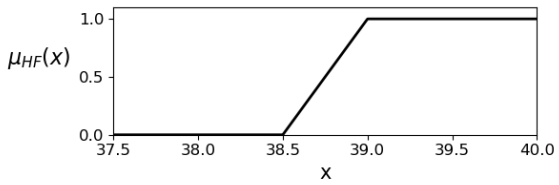

Figure 1: Graphical representation of the membership function μ_{HF} of the fuzzy set HF that represents the medical concept of "high fever".

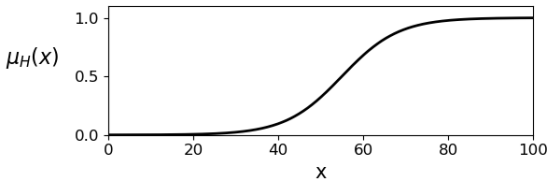

Figure 2: Graphical representation of the membership function μ_H of the fuzzy set H that represents a "developed countries in terms of health", based on the life expectancy of its population.

Definition 17 [51, p. 77]**.** A *t-conorm* (or *s-norm*) is a function $s : [0,1] \times [0,1] \to [0,1]$ that satisfies the following properties:

(i) s is commutative: $s(x,y) = s(y,x)$ for all $x, y \in [0,1]$.

(ii) s is associative: $s(x, s(y,z)) = s(s(x,y), z)$ for all $x, y, z \in [0,1]$.

(iii) 0 is the identity element: $s(x,0) = s(0,x) = x$ for all $x \in [0,1]$.

(iv) s is monotonous: if $y \leq z$, then $s(x,y) \leq s(x,z)$ for all $x, y, z \in [0,1]$.

Remark 18. Note that, due to commutativity, the property (iv) in Definition 16 is equivalent to the fact that, for all $x, y, z \in [0,1]$, if $x \leq y$, then $t(x,z) \leq t(y,z)$. The same holds analogously for the property (iv) in Definition 17. △

Let us now introduce the main operations performed on fuzzy sets. We will see that these operations are a generalization of the operations defined on classical sets.

Definition 19. Let A and B be two fuzzy sets in a crisp set X, with membership functions μ_A and μ_B, respectively. We define the *intersection* of A and B as the fuzzy set

$$A \cap B = \{(x, \mu_{A\cap B}(x)) \mid x \in X\}$$

with membership function

$$\mu_{A\cap B}(x) = t(\mu_A(x), \mu_B(x)) \quad \forall x \in X,$$

where t is a t-norm.

We define the *union* of A and B as the fuzzy set

$$A \cup B = \{(x, \mu_{A\cup B}(x)) \mid x \in X\}$$

with membership function

$$\mu_{A\cup B}(x) = s(\mu_A(x), \mu_B(x)) \quad \forall x \in X,$$

where s is an t-conorm.

And we define the *complement* of A as the fuzzy set

$$A^c = \{(x, \mu_{A^c}(x)) \mid x \in X\}$$

with membership function

$$\mu_{A^c}(x) = 1 - \mu_A(x) \quad \forall x \in X.$$

Remark 20. If the fuzzy sets A and B are crisp sets (i.e. if $\mu_A(x), \mu_B(x) \in \{0,1\}$ for all $x \in X$), then the operations on fuzzy sets become the operations on classical sets (see Definition 3).

In effect, due to 1 being the identity element of t a t-norm, it holds that $t(1,1) = 1$ and $t(1,0) = t(0,1) = 0$; and due to the monotonicity of t, we have that $t(0,0) = 0$. Therefore, the intersection of A and B satisfies the definition of the intersection of classical sets when A and B are crisp sets.

Analogously, due to 0 being the identity element of s a t-conorm, it holds that $s(0,0) = 0$ and $s(0,1) = s(1,0) = 1$; and due to s being monotonous, we have that $s(1,1) = 1$. Thus the union of A and B satisfies the definition of the union of classical sets if A and B are crisp sets.

Finally, the complement of A satisfies the definition of the complement of classical sets when A is a crisp set: $\mu_{A^c}(x) = 1 - \mu_A(x) = 1 - 1 = 0$ if x belongs to A (that is, $\mu_A(x) = 1$); and $\mu_{A^c}(x) = 1 - \mu_A(x) = 1 - 0 = 1$ if x does not belong to A (that is, $\mu_A(x) = 0$).

Therefore, the operations on fuzzy sets are a generalization of the operations on classical sets. △

We have defined the operations on fuzzy sets in terms of t-norms and t-conorms. In order to perform these operations, we need to choose specific operators (that is, specific functions that are t-norms or t-conorms). We will be interested in pairs of t-norms and t-conorms that satisfy a generalization

of De Morgan's laws for fuzzy sets, which corresponds to the notion of *duality with respect to the complement*.

Definition 21 [51, p. 83]**.** Let t be a t-norm and s be an t-conorm. We say that t and s are *dual with respect to the complement* if and only if, for any $x, y \in [0,1]$, they satisfy the following equalities:

$$t(x,y) = 1 - s(1-x, 1-y), \quad \text{and} \tag{3}$$

$$s(x,y) = 1 - t(1-x, 1-y). \tag{4}$$

Remark 22. The equalities (3) and (4) are a generalization of De Morgan's laws for fuzzy sets. In effect, consider two fuzzy sets A and B in a universal set X, with membership functions μ_A and μ_B, respectively. Let t and s be a t-norm and an t-conorm, respectively, that are dual with respect to the complement. Then, by Definitions 19 and 21, we have that

$$\mu_{A\cap B}(x) = t(\mu_A(x), \mu_B(x)) = 1 - s(1-\mu_A(x), 1-\mu_B(x)) = 1 - s(\mu_{A^c}(x), \mu_{B^c}(x)) = 1 - \mu_{A^c\cup B^c}(x),$$

and that

$$\mu_{A\cup B}(x) = s(\mu_A(x), \mu_B(x)) = 1 - t(1-\mu_A(x), 1-\mu_B(x)) = 1 - t(\mu_{A^c}(x), \mu_{B^c}(x)) = 1 - \mu_{A^c\cap B^c}(x).$$

In the case of A and B being crisp sets, these equalities correspond to De Morgan's laws for classical sets (see the equalities (1) and (2) in Remark 6). Therefore, the notion of duality with respect to the complement is a generalization of De Morgan's laws for fuzzy sets. △

Notation 23. We will denote a pair of a t-norm t and an t-conorm s that are dual with respect to the complement by $\langle t, s\rangle$. For simplicity, we will call such a pair a *dual pair* for the remainder of the text.

Commonly used dual pairs of t-norms and t-conorms

Many specific dual pairs of t-norms and t-conorms have been proposed in the literature. We now present some of the most commonly used ones [81, pp. 31, 32]. Throughout this subsection, consider $\mu_A, \mu_B : X \to [0,1]$ to be the membership functions of two fuzzy sets A, B, respectively, in a universal set X.

The minimum t-norm and maximum t-conorm[7] were proposed by L. A. Zadeh in 1965 [1]. They are defined, respectively, as

$$t_{\min}(\mu_A(x), \mu_B(y)) = \min(\mu_A(x), \mu_B(y)), \quad \text{and}$$
$$s_{\max}(\mu_A(x), \mu_B(y)) = \max(\mu_A(x), \mu_B(y)) \quad \forall\, x, y \in X.$$

In the following proposition, we prove that $t_{\min}$ and $s_{\max}$ are a dual pair.

Proposition 24. The t-norm $t_{\min}$ and the t-conorm $s_{\max}$ are dual with respect to the complement.

Proof. Let $x, y \in X$. Assume, without loss of generality, that $\mu_A(x) \leq \mu_B(y)$. Then it holds that $1 - \mu_A(x) \geq 1 - \mu_B(y)$. We obtain that

$$\begin{aligned} 1 - s_{\max}(1-\mu_A(x), 1-\mu_B(y)) &= 1 - \max(1-\mu_A(x), 1-\mu_B(y)) = 1 - (1-\mu_A(x)) = \mu_A(x) \\ &= \min(\mu_A(x), \mu_B(y)) = t_{\min}(\mu_A(x), \mu_B(y)), \quad \text{and} \\ 1 - t_{\min}(1-\mu_A(x), 1-\mu_B(y)) &= 1 - \min(1-\mu_A(x), 1-\mu_B(y)) = 1 - (1-\mu_B(y)) = \mu_B(y) \\ &= \max(\mu_A(x), \mu_B(y)) = s_{\max}(\mu_A(x), \mu_B(y)). \end{aligned}$$

Therefore, $\langle t_{\min}, s_{\max}\rangle$ is a dual pair. □

[7] The minimum t-norm and maximum t-conorm are also commonly referred to as the Gödel t-norm and Gödel t-conorm, respectively.

The Łukasiewicz t-norm and t-conorm are defined, respectively, as

$$\begin{aligned} t_{\text{Ł}}(\mu_A(x), \mu_B(y)) &= \max(0, \mu_A(x) + \mu_B(y) - 1), \quad \text{and} \\ s_{\text{Ł}}(\mu_A(x), \mu_B(y)) &= \min(1, \mu_A(x) + \mu_B(y)) \qquad\quad \forall\, x, y \in X. \end{aligned}$$

In the following proposition, we show that $t_{\text{Ł}}$ and $s_{\text{Ł}}$ are a dual pair. The proof relies on the result shown in Proposition 24.

Proposition 25. The t-norm $t_{\text{Ł}}$ and the t-conorm $s_{\text{Ł}}$ are dual with respect to the complement.

Proof. Let $x, y \in X$. Assume, without loss of generality, that $\mu_A(x) \leq \mu_B(y)$. Then it holds that $1 - \mu_A(x) \geq 1 - \mu_B(y)$.

We obtain that

$$\begin{aligned} 1 - s_{\text{Ł}}(1 - \mu_A(x), 1 - \mu_B(y)) &= 1 - \min(1,\, (1 - \mu_A(x)) + (1 - \mu_B(y))) \\ &= 1 - \min(1,\, 1 - (\mu_A(x) + \mu_B(y) - 1)) \\ &= \max(0,\, \mu_A(x) + \mu_B(y) - 1) = t_{\text{Ł}}(\mu_A(x), \mu_B(y)), \quad \text{and} \\ 1 - t_{\text{Ł}}(1 - \mu_A(x), 1 - \mu_B(y)) &= 1 - \max(0,\, (1 - \mu_A(x)) + (1 - \mu_B(y)) - 1) \\ &= 1 - \max(0,\, 1 - (\mu_A(x) + \mu_B(y))) \\ &= \min(1,\, \mu_A(x) + \mu_B(y)) = s_{\text{Ł}}(\mu_A(x), \mu_B(y)), \end{aligned}$$

where we have used that $\langle t_{\min}, s_{\max} \rangle = \langle \min, \max \rangle$ is a dual pair (Proposition 24) in the third equality of each of the equations above. Therefore, $\langle t_{\text{Ł}}, s_{\text{Ł}} \rangle$ is a dual pair. □

The last example of a dual pair of t-norm and t-conorm that we present is the product t-norm and the sum t-conorm. They are defined, respectively, as

$$t_\pi(\mu_A(x), \mu_B(y)) = \mu_A(x) \cdot \mu_B(y), \qquad \text{and} \tag{5}$$

$$s_\pi(\mu_A(x), \mu_B(y)) = \mu_A(x) + \mu_B(y) - \mu_A(x) \cdot \mu_B(y) \qquad \forall\, x, y \in X. \tag{6}$$

In order to show that t_π and s_π are a dual pair, let us first present the following two results, which are useful for finding the dual of any t-norm and any t-conorm.

Theorem 26 [51, Theorem 3.20]**.** Let t be a t-norm. Then $s : [0,1] \times [0,1] \to [0,1]$ defined by

$$s(x, y) = 1 - t(1 - x, 1 - y) \quad \forall\, x, y \in [0, 1]$$

is a t-conorm such that $\langle t, s \rangle$ is a dual pair.

Proof. Let $x, y \in [0,1]$. We start by proving that s is a t-conorm. We prove that properties (i)-(iv) in Definition 17 hold for s.

(i) s is commutative:

$$s(x, y) = 1 - t(1 - x, 1 - y) = 1 - t(1 - y, 1 - x) = s(y, x),$$

where we have used that t is commutative in the second equality.

(ii) s is associative:

$$\begin{aligned} s(x, s(y, z)) &= s(x, 1 - t(1 - y, 1 - z)) = 1 - t(1 - x, 1 - (1 - t(1 - y, 1 - z))) \\ &= 1 - t(t(1 - x, 1 - y), 1 - z) = 1 - t(1 - s(x, y), 1 - z) = s(s(x, y), z), \end{aligned}$$

where we have used the associativity of t in the third equality.

(iii) 0 is the identity element of s:

$$s(x,0) = 1 - t(1-x,1) = 1 - (1-x) = x,$$

where we have used that 1 is the identity element of t in the second equality. Similarly, by the commutativity of s, we have $s(0,x) = x$.

(iv) s is monotonous: if $y \leq z$, then $1-y \geq 1-z$, and therefore

$$s(x,y) = 1 - t(1-x,1-y) \leq 1 - t(1-x,1-z) = s(x,z),$$

where we use that $t(1-x,1-y) \geq t(1-x,1-z)$, since t is monotonous.

Therefore, s is a t-conorm. We now show that $\langle t,s\rangle$ is a dual pair. In effect,

$$\begin{aligned} &1 - s(1-x,1-y) = 1 - (1 - t(1-(1-x),1-(1-y))) = t(x,y), \quad \text{and} \\ &1 - t(1-x,1-y) = s(x,y), \quad \text{by definition.} \end{aligned}$$

□

Theorem 27 [51, Theorem 3.21]**.** Let s be a t-conorm. Then $t : [0,1]\times[0,1] \to [0,1]$ defined by

$$t(x,y) = 1 - s(1-x,1-y) \quad \forall\, x,y \in [0,1]$$

is a t-norm such that $\langle t,s\rangle$ is a dual pair.

Proof. Analogous to the proof of Theorem 26. □

We illustrate the usefulness of these results by showing that the product t-norm and sum t-conorm are a dual pair. In effect, given t_π the product t-norm defined in (5), we have that, by Theorem 26, the t-conorm s_π such that $\langle t_\pi, s_\pi\rangle$ is a dual pair is given by

$$\begin{aligned} s_\pi(\mu_A(x),\mu_B(y)) &= 1 - t_\pi(1-\mu_A(x),1-\mu_B(y)) = 1 - (1-\mu_A(x))\cdot(1-\mu_B(y)) \\ &= \mu_A(x) + \mu_B(y) - \mu_A(x)\cdot\mu_B(y) \quad \forall\, x,y \in X, \end{aligned}$$

which corresponds to the sum t-conorm as defined in (6).

We include Table 1 as a reference of the commonly used dual pairs of t-norms and t-conorms presented here, as they will be used in the following sections. The following example illustrates the use of t-norms and t-conorms to perform operations on fuzzy sets.

Example 28. Consider the universal set $X = [0,100)$. We define the fuzzy set A in X that represents "numbers considerably larger than 10" and the fuzzy set B in X that represents "numbers approximately 11". The membership functions of A and B are defined, respectively, as

$$\mu_A(x) = \begin{cases} 0 & \text{if } x \leq 10 \\ \dfrac{1}{1+(x-10)^{-2}} & \text{if } 10 < x < 100 \end{cases} \qquad \text{and} \qquad \mu_B(x) = \frac{1}{1+(x-11)^4} \qquad \forall x \in X.$$

We use the operators $\langle t_{\min}, s_{\max}\rangle$ to perform the intersection and union of A and B. The mem-

Name	***t*-norm**	***t*-conorm**
mininum / maximum	$t_{\min}(x,y) = \min(x,y)$	$s_{\max}(x,y) = \max(x,y)$
Łukasiewicz	$t_{\text{L}}(x,y) = \max(0,\, x+y-1)$	$s_{\text{L}}(x,y) = \min(1,\, x+y)$
product / sum	$t_\pi(x,y) = x\cdot y$	$s_\pi(x,y) = x+y-x\cdot y$

Table 1: Commonly used dual pairs of t-norms and t-conorms. Here we consider $x,y \in [0,1]$.

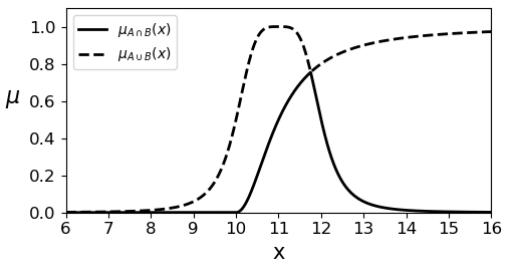

Figure 3: Graphical representation $\mu_{A\cap B}$ and $\mu_{A\cup B}$ defined in Example 28.

bership function of the fuzzy set $A \cap B$ is given by

$$\mu_{A\cap B}(x) = \begin{cases} 0 & \text{if } x \leq 10 \\ \min\left(\dfrac{1}{1+(x-10)^{-2}}, \dfrac{1}{1+(x-11)^4}\right) & \text{if } 10 < x < 100 \end{cases} = \begin{cases} 0 & \text{if } x \leq 10 \\ \dfrac{1}{1+(x-10)^{-2}} & \text{if } 10 < x \leq x_0 \\ \dfrac{1}{1+(x-11)^4} & \text{if } x_0 < x < 100 \end{cases}$$

and the membership function of the fuzzy set $A \cup B$ is given by

$$\mu_{A\cup B}(x) = \begin{cases} \dfrac{1}{1+(x-11)^4} & \text{if } x \leq 10 \\ \max\left(\dfrac{1}{1+(x-10)^{-2}}, \dfrac{1}{1+(x-11)^4}\right) & \text{if } 10 < x < 100 \end{cases} = \begin{cases} \dfrac{1}{1+(x-11)^4} & \text{if } 0 \leq x \leq x_0 \\ \dfrac{1}{1+(x-10)^{-2}} & \text{if } x_0 < x < 100, \end{cases}$$

where x_0 is the point where the membership functions μ_A and μ_B intersect. Solving the equation $\mu_A(x_0) = \mu_B(x_0)$ numerically for x_0 yields $x_0 \approx 11.7549$. See Figure 3 for a graphical representation of $\mu_{A\cap B}$ and $\mu_{A\cup B}$. △

Interpretation of fuzzy sets and their operations in fuzzy logic

Let X be a universal set. Recall, from section 2.1, that a classical logic proposition $P(x)$, defined for every $x \in X$, determines a crisp set $A = \{x \in X \mid P(x)\}$. This connection is extended to fuzzy logic propositions and fuzzy sets: a fuzzy logic proposition[8] $P(x)$, defined for every $x \in X$, determines a fuzzy set $A = \{(x, \mu_A(x)) \mid x \in X\}$ such that $\mu_A(x)$ is the degree of truth of $P(x)$ $\forall x \in X$ [51, p. 220].

Furthermore, the operations of intersection, union and complement of fuzzy sets are interpreted as the logical operations of conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$) of fuzzy logic propositions, respectively [82]. For instance, if $P(x)$ and $Q(x)$ are two fuzzy logic propositions, defined for all $x \in X$, with corresponding fuzzy sets $A = \{(x, \mu_A(x)) \mid x \in X\}$ and $B = \{(x, \mu_B(x)) \mid x \in X\}$, respectively, then the degree of truth of $P(x) \wedge Q(x)$ is given by $\mu_{A\cap B}(x)$. The same relationship holds analogously for $P(x) \vee Q(x)$ and $\mu_{A\cup B}(x)$, and for $\neg P(x)$ and $\mu_{A^c}(x)$.

Example 29. Consider the universal set $X = [0, 100)$. Let $P(x)$ and $Q(x)$ be the fuzzy logic propositions "x is considerably larger than 10" and "x is approximately 11", respectively, defined for all $x \in X$. These propositions correspond to the fuzzy sets A and B defined in Example 28. Consider also $\mu_{A\cap B}$ and $\mu_{A\cup B}$ as defined in Example 28.

Let $x = 11.4$. The truth values of the propositions $P(11.4)$ and $Q(11.4)$ are given by $\mu_A(11.4) \approx 0.6622$ and $\mu_B(11.4) \approx 0.9750$, respectively. Furthermore, the degrees of truth of propositions $P(11.4) \wedge Q(11.4)$, $P(11.4) \vee Q(11.4)$ and $\neg P(11.4)$ are given by $\mu_{A\cap B}(11.4) \approx 0.6622$, $\mu_{A\cup B}(11.4) \approx 0.9750$ and $\mu_{A^c}(11.4) = 1 - \mu_A(11.4) \approx 0.3378$, respectively. △

Remark 30. We have shown in Remark 20 that operations on fuzzy sets are a generalization of the corresponding operations on crisp sets. This fact ensures that the fuzzy logical operators $\wedge$, $\vee$ and $\neg$ preserve the corresponding operations in classical propositional logic.

In effect, the operations of conjunction, disjunction and negation of the Boolean algebra on $\{0, 1\}$

[8] As described in the Introduction of the present text (section 1), fuzzy logic propositions typically represent vague predicates which are satisfied to a certain degree, such as "the building x is tall", "the product x is cheap", or "the painting x is an Impressionist painting". This differs from the kind of predicates that are typically represented by classical logic propositions, which accept binary *true* or *false* values, such as "the number x is prime", "the person x is alive", or "the phrase x is a palindrome".

are preserved [77; 51, ch. 8]:

$$
\begin{array}{lll}
0 \wedge 0 = t(0,0) = 0 & 0 \vee 0 = s(0,0) = 0 & \neg 0 = c(0) = 1 \\
0 \wedge 1 = t(0,1) = 0 & 0 \vee 1 = s(0,1) = 1 & \neg 1 = c(1) = 0 \\
1 \wedge 0 = t(1,0) = 0 & 1 \vee 0 = s(1,0) = 1 & \\
1 \wedge 1 = t(1,1) = 1 & 1 \vee 1 = s(1,1) = 1 &
\end{array}
$$

where t is a t-norm and s is a t-conorm such that $\langle t, s \rangle$ is a dual pair, and $c(x) = 1 - x$ denotes the fuzzy complement operation. △

For an in-depth study of the mathematical foundations of fuzzy logic, the reader is referred to [83].

2.4 Application: design and implementation of a fuzzy rule-based system

An early use case of fuzzy set theory and fuzzy logic in the field of Artificial Intelligence has been the design of *fuzzy rule-based systems* (also commonly referred to as *fuzzy inference systems*). The goal of these systems is to make predictions or decisions based on input data and a set of fuzzy logic rules, using the framework provided by fuzzy set theory. One application of such systems which appears frequently in the literature is the prediction of crop yield, that is, the amount of a crop that is produced on a given area of land in a single growing season.

In this section, we construct a fuzzy rule-based system to predict the crop yield of rice plants[9]. According to [85], two of the most important factors that determine the crop yield (Y) of rice plant varieties are the panicle number per area (P) and the growth period (G) of the plants[10,11]. We develop a fuzzy rule-based system to predict Y based on P and G. The system will be implemented and tested in Python using a dataset of 534 samples of rice plants, for which the values of P, G and Y are known. The dataset is obtained from [85].

In order to construct the fuzzy rule-based system, the dataset has to be previously processed. Once the dataset is processed, four steps are followed for the design of the system [51]:

(i) fuzzy sets are defined for each of the variables involved in the system (P, G and Y),

(ii) the input variables (P and G) are *fuzzified*, that is, for each input variable, we calculate the membership value for each of the corresponding fuzzy sets,

(iii) a set of rules (*fuzzy inference rules*) are defined and evaluated to infer the membership value of each fuzzy set associated to the output variable (thus obtaining the *fuzzy output*), and

(iv) the fuzzy output is *defuzzified*, that is, it is transformed into the crisp predicted value for Y.

Figure 4 depicts the scheme of a fuzzy rule-based system. We discuss each of these steps in the following subsections. Afterward, we discuss the results of the implementation of the system.

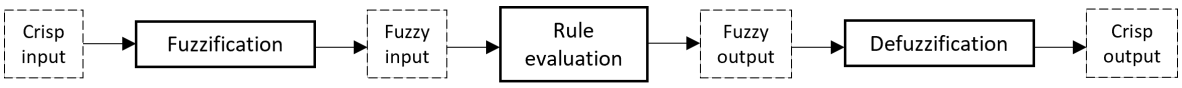

Figure 4: Scheme of a fuzzy rule-based system.

[9]The idea for this fuzzy rule-based system is based on [84], where it was implemented for the prediction of crop yields of Sorghum plants. The system presented here is adapted to yields of rice plants, and it is based on the statistical analysis performed in [85]. Furthermore, here we present a simplified version with respect to the one described in [84], with the purpose of illustrating this early application of fuzzy set theory. In particular, we consider only two input variables (panicle number and growth period) to predict the yield, and less fuzzy sets for each variable.

[10] It is shown in [85] that the factors that influence the crop yield of rice plant varieties vary depending on the type of rice plant. We focus here on the Japonica inbred type of rice plants, for which P and G are shown in [85] to be the most significant factors that influence Y.

[11]Crop yield (Y) is defined here as the amount of crop produced per unit of area, measured in tons per hectare (t/ha). The panicle number per area (P) is defined as the number of panicles of the crop per unit of area, measured in millions of panicles per hectare (million/ha). The growth period (G) of the crop, defined as the number of days between the sowing of the seeds and the harvest of the crop, is measured in days (d).

Description of the dataset and data processing

The dataset used to implement the system consists of 534 samples corresponding to different varieties of rice plants, and 3 variables: the panicle number per area (P), the growth period (G) and the crop yield (Y). Table 2 shows an excerpt of the dataset. Columns `panicle`, `growth` and `yield` correspond to variables P, G and Y, respectively.

sample ID	panicle	growth	yield
1	3.975	184.0	6.7500
2	2.850	125.0	6.4275
3	4.500	147.5	7.2285
4	5.100	167.5	8.7150
5	3.675	180.0	7.7820

Table 2: Excerpt of the dataset used to implement the fuzzy rule-based system.

The columns `panicle` and `growth`, corresponding to the input variables P and G, are used by the system to predict the output variable Y. The values in the column `yield` are the real values of Y for each sample, and will be used to test the performance of the system by comparing the predicted and the real values for Y.

The data, obtained from [85], has been processed here in the following way: (i) the samples of the Japonica inbred type of rice plants have been selected (see footnote 10), (ii) the samples with missing values have been removed, and (iii) for each column, outliers have been removed[12]. The resulting dataset consists of 534 samples of rice plants, with no missing values or outliers. Table 3 shows the description of the dataset, including common statistical indicators.

	panicle	growth	yield
count	534.0	534.0	534.0
mean	3.4708	153.4152	8.8879
std	0.5527	14.8409	0.8978
min	2.55	123.4	6.4011
10%	2.895	135.0	7.713810
25%	3.0638	142.0	8.4034
50%	3.3075	154.0	8.8557
75%	3.75	160.725	9.4193
90%	4.296	175.97	10.0995
max	5.115	191.5	11.19

Table 3: Description of the dataset used for the prediction of crop yields. The statistical indicators `std`, `10%`, `25%`, `50%`, `75%` and `90%` correspond to the standard deviation, and the 10th, 25th, 50th, 75th and 90th percentiles, respectively. The values are rounded to four decimal places.

The processing of the data has been done using the Pandas library in the Python 3 programming language. The code used for the data processing and the resulting dataset is available in the Github repository found online at: https://github.com/loredanasandu/tfg-fuzzy-logic-artificial-intelligence. An extract with the relevant parts of the code is included in Appendix A, section A.1.

Having processed the dataset, we now proceed to implement the fuzzy rule-based system. We start by defining the fuzzy sets for each of the variables.

Definition of the fuzzy sets

For each of the input variables (P and G) and the output variable (Y), we define five fuzzy sets, which represent the linguistic terms "very low" (VL), "low" (L), "medium" (M), "high" (H) and "very high" (VH). The notation used for every fuzzy set is specified in Table 4. The membership functions of these fuzzy sets are defined in Table 5 and represented in Figure 5.

	P	**G**	**Y**
Very low	VLP	VLG	VLY
Low	LP	LG	LY
Medium	MP	MG	MY
High	HP	HG	HY
Very high	VHP	VHG	VHY

Table 4: Notation for the fuzzy sets considered in the fuzzy rule-based system.

To define the membership functions (namely the bounds of the intervals of the piece-wise functions), the 10th, 25th, 50th, 75th and 90th percentiles of each column in the dataset were used (Table 3). Also, the membership functions are defined for the range of the data (that is, between the minimum and maximum values of the corresponding column in the dataset). For example, considering the definition of μ_{VLY} in Table 5, the values 7.71, 8.4, 6.4 and 11.19 are the 10th percentile, the 25th percentile, the minimum and the maximum of the corresponding column `yield` in the dataset, respectively. The same procedure was used to define the rest of membership functions.

[12]More specifically,we remove the samples for which any feature is outside the range of (Q_1, Q_{99}), where Q_1 and Q_{99} are the 1st and 99th percentiles of the data for the corresponding feature, respectively. Removing the outliers is a common practice when implementing artificial intelligence systems, primarily machine learning and deep learning models [39]. It is done in order to avoid the influence of outliers in the training of the models, this way improving the accuracy of the predictions.

	P	G	Y
VL	$\mu_{VLP}(x) = \begin{cases} 1 & \text{if } x < 2.9 \\ \frac{3.07-x}{0.17} & \text{if } 2.9 \leq x \leq 3.07 \\ 0 & \text{if } x > 3.07 \end{cases}$	$\mu_{VLG}(x) = \begin{cases} 1 & \text{if } x < 135.0 \\ \frac{142.0-x}{7.0} & \text{if } 135.0 \leq x \leq 142.0 \\ 0 & \text{if } x > 142.0 \end{cases}$	$\mu_{VLY}(x) = \begin{cases} 1 & \text{if } x < 7.71 \\ \frac{8.4-x}{0.7} & \text{if } 7.71 \leq x \leq 8.4 \\ 0 & \text{if } x > 8.4 \end{cases}$
L	$\mu_{LP}(x) = \begin{cases} \frac{x-2.9}{0.17} & \text{if } 2.9 \leq x \leq 3.07 \\ \frac{3.31-x}{0.24} & \text{if } 3.07 \leq x \leq 3.31 \\ 0 & \text{if } x < 2.9 \text{ or } 3.31 < x \end{cases}$	$\mu_{LG}(x) = \begin{cases} \frac{x-135.0}{7.0} & \text{if } 135.0 \leq x \leq 142.0 \\ \frac{154.0-x}{12.0} & \text{if } 142.0 \leq x \leq 154.0 \\ 0 & \text{if } x < 135.0 \text{ or } 154.0 < x \end{cases}$	$\mu_{LY}(x) = \begin{cases} \frac{x-7.71}{0.7} & \text{if } 7.71 \leq x \leq 8.4 \\ \frac{8.86-x}{0.46} & \text{if } 8.4 \leq x \leq 8.86 \\ 0 & \text{if } x < 7.71 \text{ or } 8.86 < x \end{cases}$
M	$\mu_{MP}(x) = \begin{cases} \frac{x-3.07}{0.24} & \text{if } 3.07 \leq x \leq 3.31 \\ \frac{3.75-x}{0.44} & \text{if } 3.31 \leq x \leq 3.75 \\ 0 & \text{if } x < 3.07 \text{ or } 3.75 < x \end{cases}$	$\mu_{MG}(x) = \begin{cases} \frac{x-142.0}{12.0} & \text{if } 142.0 \leq x \leq 154.0 \\ \frac{160.7-x}{6.7} & \text{if } 154.0 \leq x \leq 160.7 \\ 0 & \text{if } x < 142.0 \text{ or } 160.7 < x \end{cases}$	$\mu_{MY}(x) = \begin{cases} \frac{x-8.4}{0.46} & \text{if } 8.4 \leq x \leq 8.86 \\ \frac{9.42-x}{0.56} & \text{if } 8.86 \leq x \leq 9.42 \\ 0 & \text{if } x < 8.4 \text{ or } 9.42 < x \end{cases}$
H	$\mu_{HP}(x) = \begin{cases} \frac{x-3.31}{0.44} & \text{if } 3.31 \leq x \leq 3.75 \\ \frac{4.3-x}{0.55} & \text{if } 3.75 \leq x \leq 4.3 \\ 0 & \text{if } x < 3.31 \text{ or } 4.3 < x \end{cases}$	$\mu_{HG}(x) = \begin{cases} \frac{x-154.0}{6.7} & \text{if } 154.0 \leq x \leq 160.7 \\ \frac{175.97-x}{15.27} & \text{if } 160.7 \leq x \leq 175.97 \\ 0 & \text{if } x < 154.0 \text{ or } 175.97 < x \end{cases}$	$\mu_{HY}(x) = \begin{cases} \frac{x-8.86}{0.56} & \text{if } 8.86 \leq x \leq 9.42 \\ \frac{10.1-x}{0.68} & \text{if } 9.42 \leq x \leq 10.1 \\ 0 & \text{if } x < 8.86 \text{ or } 10.1 < x \end{cases}$
VH	$\mu_{VHP}(x) = \begin{cases} 0 & \text{if } x < 3.75 \\ \frac{x-3.75}{0.55} & \text{if } 3.75 \leq x \leq 4.3 \\ 1 & \text{if } x > 4.3 \end{cases}$	$\mu_{VHG}(x) = \begin{cases} 0 & \text{if } x < 160.7 \\ \frac{x-160.7}{15.27} & \text{if } 160.7 \leq x \leq 175.97 \\ 1 & \text{if } x > 175.97 \end{cases}$	$\mu_{VHY}(x) = \begin{cases} 0 & \text{if } x < 9.42 \\ \frac{x-9.42}{0.68} & \text{if } 9.42 \leq x \leq 10.1 \\ 1 & \text{if } x > 10.1 \end{cases}$

Table 5: Membership functions for the fuzzy sets considered in the fuzzy rule-based system. The membership functions for P are defined for all real numbers $x \in [2.55, 5.12]$, the membership functions for G are defined for all $x \in [123.4, 191.5]$, and the membership functions for Y are defined for all $x \in [6.4, 11.19]$.

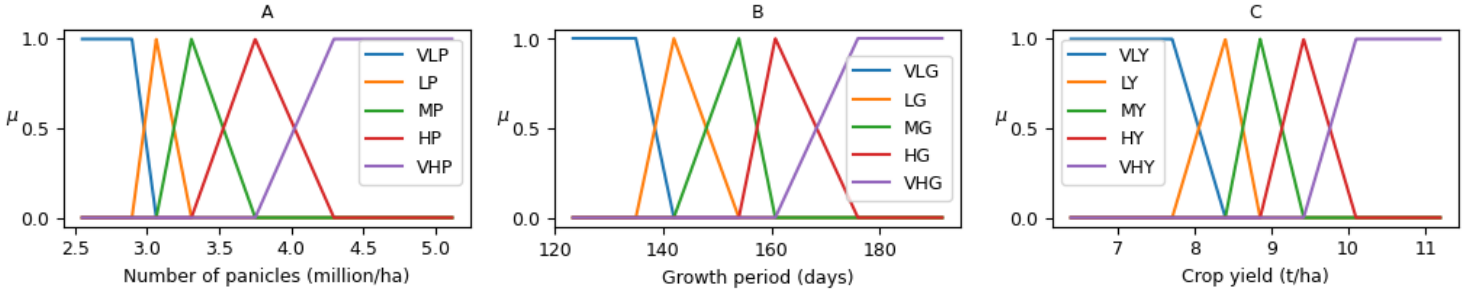

Figure 5: Graphical representation of the membership functions of the fuzzy sets defined for variable P (A), variable G (B) and variable Y (C).

Fuzzification of the input variables

We now fuzzify the input variables P and G. For each of these variables, we calculate the membership value of each of the corresponding fuzzy sets.

As an example of the result of the fuzzification process, consider the value $x = 147.5$ for the variable G. The membership values of x for each of the fuzzy sets associated to G are given by

$$\mu_{LG}(147.5) = \frac{154.0 - 147.5}{12.0} \approx 0.5417, \quad \mu_{MG}(147.5) = \frac{147.5 - 142.0}{12.0} \approx 0.4583,$$
$$\mu_{VLG}(147.5) = \mu_{HG}(147.5) = \mu_{VHG}(147.5) = 0.0.$$

Table 6 shows the result of the fuzzification of variables P and G for the samples shown in the excerpt of the dataset in Table 2.

sample ID	VLP	LP	MP	HP	VHP	VLG	LG	MG	HG	VHG
1	0.0	0.0	0.0	0.5879	0.4121	0.0	0.0	0.0	0.0	1.0
2	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	1.0	0.0	0.5417	0.4583	0.0	0.0
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.5556	0.4444
5	0.0	0.0	0.1695	0.8305	0.0	0.0	0.0	0.0	0.0	1.0

Table 6: Result of the fuzzification process for an excerpt of the dataset used to implement the fuzzy rule-based system. The second to last column correspond to the membership values of the fuzzy sets for the input. The values are rounded to four decimal places.

Definition and evaluation of the fuzzy logic rules

We now proceed to define the fuzzy logic rules, and evaluate them in order to infer the membership value of the fuzzy sets corresponding to the output variable (Y). The rule base is composed of 25 rules in the form of fuzzy logic statements. The rules considered here follow the following format:

IF P is `<fuzzy set for P>` AND G is `<fuzzy set for G>` THEN Y is `<fuzzy set for Y>`

For each combination of fuzzy sets for the variables P and G, there is a rule that determines the fuzzy set for the output variable Y and its inferred (predicted) membership value, by performing the $\wedge$ operation on the membership values of the antecedent fuzzy sets. For example, the rule

IF P is LP AND G is MG THEN Y is LY

infers the membership value of the fuzzy set LY, by performing the $\wedge$ operation on the membership values of the antecedent fuzzy sets LP and MG:

$$p_{LY} = \mu_{LP}(x_1) \wedge \mu_{MG}(x_2) = t(\mu_{LP}(x_1), \mu_{MG}(x_2)),$$

where p_{LY} denotes the membership value predicted by this rule for the consequent fuzzy set LY, t is a t-norm, and x_1 and x_2 are the given values of variables P and G, respectively, for a given sample in the dataset.

Table 7 shows the specific fuzzy sets considered in each of the 25 rules[13]. All rules are evaluated for each sample in the dataset. If, for any given sample, multiple membership values for the same fuzzy set result from different rules, we aggregate them by performing the $\vee$ operation on the different membership values of the consequent fuzzy set. For example, if the rules

IF P is VHP AND G is HG THEN Y is VHY

IF P is VHP AND G is VHG THEN Y is VHY

	VLP	**LP**	**MP**	**HP**	**VHP**
VLG	VLY	VLY	LY	LY	MY
LG	LY	LY	MY	MY	MY
MG	LY	LY	MY	MY	HY
HG	MY	MY	HY	HY	VHY
VHG	MY	MY	HY	HY	VHY

Table 7: Rule base considered for the fuzzy rule-based system. The column names and row names correspond to the fuzzy sets considered in the antecedent of each rule. The values in the table correspond to the resulting fuzzy set, in the consequent of each rule.

are evaluated given the sample with values $x = 5.1$ for P and $x = 167.5$ for G (sample 4 in the excerpt of the dataset in Table 6) using the $\langle t_{\min}, s_{\max}\rangle$ dual pair, the membership values for VHY resulting from each rule are given by

$$p^1_{VHY} = \mu_{VHP}(5.1) \wedge \mu_{HG}(167.5) = t_{\min}(\mu_{VHP}(5.1), \mu_{HG}(167.5)) = t_{\min}(1.0, 0.5556) = 0.5556$$
$$p^2_{VHY} = \mu_{VHP}(5.1) \wedge \mu_{MG}(167.5) = t_{\min}(\mu_{VHP}(5.1), \mu_{MG}(167.5)) = t_{\min}(1.0, 0.4444) = 0.4444$$

The predicted membership value of the fuzzy set VHY is then given by

$$p_{VHY} = p^1_{VHY} \vee p^2_{VHY} = s_{\max}(p^1_{VHY}, p^2_{VHY}) = s_{\max}(0.5556, 0.4444) = 0.5556.$$

Table 8 shows the result of the evaluation of the rules for the samples of the excerpt of the dataset shown in Table 2. The values in the table correspond to the membership values for each fuzzy set for Y, predicted by the system. The evaluation shown in the table has been performed using the $\langle t_{\min}, s_{\max}\rangle$ dual pair. The system will be tested using different dual pairs.

[13]Using results from [85], many different rule bases have been tested. The rule base considered here has shown the best performance for predicting the crop yield with the input data used in this example, using different dual pairs. In subsequent section, methods to generate the rule base automatically, combining neural networks and fuzzy set theory, will be introduced and tested.

sample ID	predicted VLY	predicted LY	predicted MY	predicted HY	predicted VHY
1	0.0	0.0	0.0	0.5879	0.4121
2	1.0	0.0	0.0	0.0	0.0
3	0.0	0.5417	0.4583	0.0	0.0
4	0.0	0.0	0.0	0.0	0.5556
5	0.0	0.0	0.0	1.0	0.0

Table 8: Result of the evaluation of the rules for an excerpt of the dataset used to implement the fuzzy rule-based system. The values in the table correspond to the predicted membership values of each fuzzy set for Y. The evaluation shown here has been performed using the $\langle t_{\min}, s_{\max}\rangle$ dual pair.

Defuzzification of the output

Having evaluated the rules, the last step is to defuzzify the output. We transform the predicted membership values for the output fuzzy sets into a crisp value for Y, which is the predicted crop yield resulting from the system. We will use the *centroid method* to defuzzify the output[14].

The centroid method of defuzzification for variable Y is defined as follows (an example is provided below). Consider the fuzzy set VLY, with membership function μ_{VLY} (defined in Table 5), and let p_{VLY} be the membership value predicted by the system for this fuzzy set. We define the function

$$\mu^p_{VLY}(x) := \min\left(\mu_{VLY}(x), p_{VLY}\right), \quad \forall x \in [6.4, 11.19],$$

Define $\mu^p_{LY}(x)$, $\mu^p_{MY}(x)$, $\mu^p_{HY}(x)$ and $\mu^p_{VHY}(x)$ analogously. Next, consider the function

$$\mu^p_Y(x) := \max\left(\mu^p_{VLY}(x), \mu^p_{LY}(x), \mu^p_{MY}(x), \mu^p_{HY}(x), \mu^p_{VHY}(x)\right), \quad \forall x \in [6.4, 11.19].$$

The centroid method of defuzzification returns the center of gravity of the area under the curve of the function $\mu^p_Y(x)$, that is, the point $x_c \in [6.4, 11.19]$ such that the area under the curve of $\mu^p_Y(x)$ between 6.4 and x_c is equal to the area under the curve of $\mu^p_Y(x)$ between x_c and 11.19. The value of x_c is given by

$$x_c = \frac{\int_{6.4}^{11.19} x \cdot \mu^p_Y(x) dx}{\int_{6.4}^{11.19} \mu^p_Y(x) dx},$$

which is the predicted crop yield resulting from the system.

As an example, consider the sample with values $x = 4.5$ for P and $x = 147.5$ for G. We have seen previously that the membership values for the fuzzy sets LY and MY predicted by the system are $p_{LY} = 0.5417$ and $p_{MY} = 0.4583$, respectively, and 0.0 for the rest of the fuzzy sets (sample 3 in Table 8). We plot the functions μ^p_{LY}, μ^p_{MY} and μ^p_Y, obtained computationally, in Figure 6. The center of gravity of the area under $\mu^p_Y(x)$ is given by $x_c \approx 9.2062$ (approximated computationally). Thus, the predicted crop yield for the sample with values $x = 4.5$ for P and $x = 147.5$ for G is 9.2062.

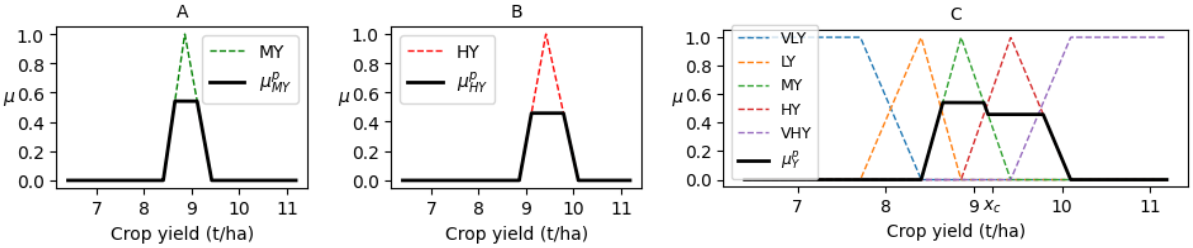

Figure 6: Graphical representation of the functions μ^p_{MY} (A), μ^p_{HY} (B) and μ^p_Y (C) for the sample with values $x = 4.5$ for P and $x = 147.5$ for G. Dashed lines represent the membership functions of the fuzzy sets for Y. The center of gravity of the area under the curve of μ^p_Y is given by $x_c \approx 9.2062$.

[14]There are other defuzzification methods, such as the *center of maximum*, which takes the middle value of the α-cut for the set with the maximum predicted membership value and for α the predicted membership value. Although it is computationally more efficient, this method is normally less accurate than the centroid method [81], and it is not used here.

Table 9 shows the predicted crop yield for each sample of the excerpt of the dataset, obtained with the centroid method of defuzzification.

sample ID	predicted yield
1	10.0136
2	7.2406
3	9.2062
4	10.3966
5	9.458154

Table 9: Predicted crop yield for each sample for an excerpt of the dataset, rounded to four decimal places. The result is based on the predicted membership values shown in Table 8.

We now proceed to test the performance of the system with all the samples in the dataset, and using different dual pairs.

Results

We apply the fuzzy rule-based system to all the samples in the dataset. We use the $\langle t_{\min}, s_{\max}\rangle$, $\langle t_{\text{L}}, s_{\text{L}}\rangle$ and $\langle t_{\pi}, s_{\pi}\rangle$ dual pairs, introduced in section 2.3. To measure the performance of the system, we use the Root Mean Square Error (RMSE), which is defined as follows:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}, \tag{7}$$

where N is the number of samples in the dataset (in our case, $N = 534$), y_i is the real value of the crop yield for the i-th sample, and $\hat{y}_i$ is the predicted value of the crop yield for the i-th sample. The RMSE is a measure of the differences between the values predicted by the system and the real values of the crop yield. The lower the RMSE, the better the performance of the system. Table 10 shows the RMSE obtained for the prediction of crop yields for all samples in the dataset, using the $\langle t_{\min}, s_{\max}\rangle$, $\langle t_{\text{L}}, s_{\text{L}}\rangle$ and $\langle t_{\pi}, s_{\pi}\rangle$ dual pairs.

Dual pair	**RMSE**
$\langle t_{\min}, s_{\max}\rangle$	0.8595
$\langle t_{\text{L}}, s_{\text{L}}\rangle$	0.8676
$\langle t_{\pi}, s_{\pi}\rangle$	0.8606

Table 10: RMSE for the prediction of the crop yield using the fuzzy rule-based system, for each dual pair. The RMSE is rounded to four decimal places.

Although there is no significant difference in performance among the three dual pairs, we notice that $\langle t_{\min}, s_{\max}\rangle$ shows the best performance, and $\langle t_{\text{L}}, s_{\text{L}}\rangle$ shows the worst performance. Differences in performance with respect to dual pairs will also be observed in the results of other predictive models using fuzzy logic, such as *fuzzy neural networks*, which will be presented in subsequent sections.

A plot of the predicted and the real crop yields for samples 50 to 450 in the dataset is included in Figure 7. The plot shows that the system is able to predict the changes in crop yield among different rice plant varieties with a notable degree of accuracy for some samples. However, the prediction is not accurate for all samples, nor for a large number of them. This should be taken into account before generalizing the rules and methods used here to new data, or to different contexts.

The dataset and code for the implementation and testing of the fuzzy rule-based system is available in the Github repository found online at: https://github.com/loredanasandu/tfg-fuzzy-logic-artificial-intelligence. An extract with the relevant parts of the code is included in Appendix A, section A.2.

The reader interested in delving deeper into the subject of fuzzy set theory and its applications (including the design and implementation of fuzzy rule-based systems) is referred to [51, 81].

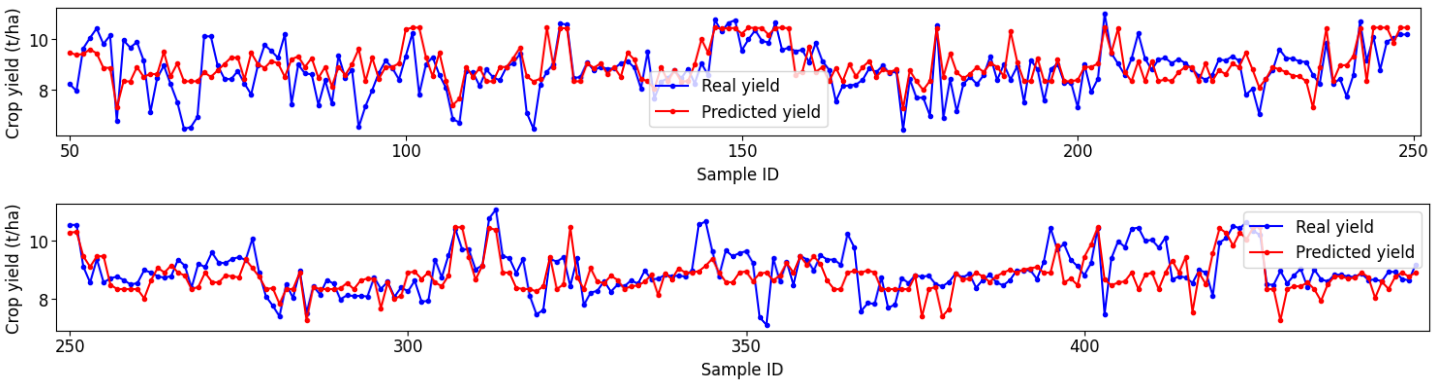

Figure 7: Real and predicted crop yields for samples 50 to 450 in the dataset, using the fuzzy rule-based system with $\langle t_{\pi}, s_{\pi}\rangle$.

3 Fuzzy neural networks

The aim of this section is to present the mathematical model known as *fuzzy neural network*. This model integrates fuzzy logic operators into the classical structure of neural networks. The new structure resulting from this approach is shown to reveal underlying logical relationships between the input and output variables, which cannot be explicitly extracted from classical neural networks. Therefore, fuzzy neural networks are able to generate more interpretable results than classical neural networks.

We start by introducing the fundamental notions related to neural networks, which are then extended to fuzzy neural networks. We also demonstrate the application of these models to the prediction of crop yields, and compare the results to those obtained with the fuzzy rule-based system presented in the previous section.

3.1 Neural networks: an overview

A neural network is a mathematical model inspired by the biological neural networks found in the human brain. It is designed to learn and approximate complex relationships between input and output data by adjusting its internal parameters through a training process.

The architecture of a neural network[15] typically consists of multiple layers, including an input layer, one or more hidden layers, and an output layer. Each layer contains a set of neurons, which are interconnected by weighted connections. A classical neural network can be represented by a directed graph like the one presented in Figure 8.

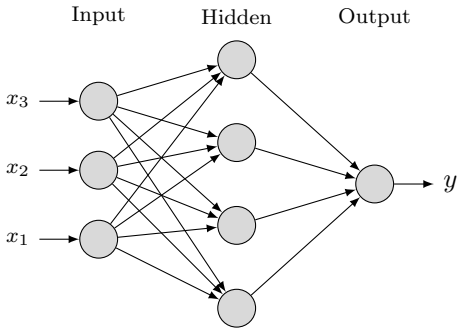

Figure 8: An example feed-forward neural network with input vector $x = (x_1, x_2, x_3)$, one hidden layer with four neurons and one output scalar y.

Let us now proceed with the formal description of a neural network.

A *neuron* is a function of the form

$$y(\mathbf{x}) = a(\mathbf{x} \cdot \mathbf{w} + b) = a\left(\sum_{i=1}^{n} w_i \cdot x_i + b\right)$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ is the input vector, $\mathbf{w} = (w_1, \ldots, w_n)$ is the weight vector, b is the bias, and a is an *activation function*. The weight vector w and the bias b are the *parameters* of the neuron. The activation function a is typically a non-linear function that transforms its input into a bounded output. We include a list of some of the most commonly used activation functions in Table 11. The output y of the neuron is the scalar resulting from applying the activation function a to the weighted sum of the inputs plus the bias.

Activation function	**Description**
Rectified linear unit (ReLU)	$f : \mathbb{R} \to [0, +\infty) \subset \mathbb{R}$ such that $f(x) = \max(0, x)$
Sigmoid	$f : \mathbb{R} \to (0, 1) \subset \mathbb{R}$ such that $f(x) = \frac{1}{1+e^{-x}}$
Hyperbolic tangent	$f : \mathbb{R} \to (-1, 1) \subset \mathbb{R}$ such that $f(x) = \tanh(x)$

Table 11: List of some of the most commonly used activation functions. For further details these and other activation functions, including results regarding their performance, the reader is referred to [39, 87].

[15] Though there are multiple architectures of neural networks in the literature, we will focus here on feed-forward neural networks, also commonly referred to as *multilayer perceptrons* (MLP). The description of this model provided here is based on the approach presented in [39, 86].

A *neural network* (NN) is a composition of neurons. It typically consists of an input layer, one or more hidden layers, and an output layer. Each layer contains a set of neurons, and the neurons in each layer are connected to the neurons in the next layer in the sense that the outputs of the neurons in one layer are the inputs of the neurons in the next layer. The input of the neural network is the input of the first layer in the network, and its output is the output of the last layer (see Figure 8 for a graphical representation).

More precisely, consider a neural network with $L+1$ layers. Let $n_0, \dots, n_L$ be the number of neurons in each layer, and let $a_0, \dots, a_L$ be the activation functions of each layer, where the 0-th layer denotes the input layer, and the L-th layer denotes the output layer. The output of the k-th neuron in the l-th layer is given by

$$z_k^{(l)}(\mathbf{x}) = a_l\left(\sum_{i=1}^{n_{l-1}} w_{ki}^{(l)} z_i^{(l-1)}(\mathbf{x}) + b_k^{(l)}\right),$$

where $\mathbf{x}$ is the input vector of the neural network, $z_i^{(l-1)}(\mathbf{x})$ is the output of the i-th neuron in the $(l-1)$-th layer, $w_{ki}^{(l)}$ is the weight of the connection between the i-th neuron in the $(l-1)$-th layer and the k-th neuron in the l-th layer, and $b_k^{(l)}$ is the bias of the k-th neuron in the l-th layer, with $l \in \{1, \dots, L\}$ and $k \in \{1, \dots, n_l\}$. The output of the i-th neuron in the 0-th layer is $z_i^{(0)}(\mathbf{x}) = x_i$, where x_i is the i-th component of the input vector $\mathbf{x}$.

The vector of outputs of the neurons at the l-th layer is

$$\mathbf{z}^{(l)}(\mathbf{x}) = a_l\Big(\mathbf{W}^{(l)}\mathbf{z}^{(l-1)}(\mathbf{x}) + \mathbf{b}^{(l)}\Big) =: f_l\big(\mathbf{z}^{(l-1)}(\mathbf{x})\big),$$

where $\mathbf{W}^{(l)} = \left(w_{ki}^{(l)}\right)_{k,i=1}^{n_l, n_{l-1}}$ is the matrix of weights of the connections between the neurons in the $(l-1)$-th layer and the neurons in the l-th layer, $\mathbf{b}^{(l)} = \left(b_k^{(l)}\right)_{k=1}^{n_l}$ is the vector of biases of the neurons in the l-th layer, and $\mathbf{z}^{(l-1)}(\mathbf{x})$ is the vector of outputs of the neurons in the $(l-1)$-th layer, for $l \in \{1, \dots, L\}$. The vector $\mathbf{z}^{(0)}(\mathbf{x})$ of outputs of the neurons at the 0-th layer is $\mathbf{z}^{(0)}(\mathbf{x}) = \mathbf{x}$.

The neural network is the composition of f_l for $l = 1, \dots, L$, that is, the function

$$\begin{aligned} F : \mathbb{R}^{n_0} &\longrightarrow \mathbb{R}^{n_L} \\ \mathbf{x} &\longmapsto f_L\big(f_{L-1}\big(\cdots f_1(\mathbf{x})\cdots\big)\big) \end{aligned}$$

Note that, if the activation functions are chosen to be linear functions, then the neural network is a linear function, since the composition of linear functions is a linear function. Therefore, non-linear activation functions are typically chosen, in order for the neural network to be able to approximate non-linear functions[16].

The *training process* of a neural network consists of a series of iterations over input data with known output, where the objective is to minimize the error between the output of the neural network (the *predicted output*) and the known output. In each iteration, the parameters of the neural network (namely the weights w and the biases b in the notation above) are adjusted to minimize this error[17], which is typically measured using a *loss function*. We include a list of some of the most commonly used loss functions in Table 12. In each iteration, the neural network is expected to improve its performance, that is, to decrease the value of the loss function further. The training process is typically stopped when the loss function reaches a minimum, or when the performance of the neural network stops improving significantly.

[16] Specific choices of activation functions depend on the application of the neural network. The ReLU function has been shown to perform well for many use cases [87], and is one of the most commonly used in hidden layers. Regarding the output layer, one should consider the range of the output variables, among other factors. One of the most commonly used activation functions in the output layer is the sigmoid function, since it returns a real value in $(0, 1)$, which can be interpreted as a probability (for example, in classification problems), or be easily rescaled to adapt to other cases.

[17] The optimization of the parameters is typically performed using variants of the gradient descent algorithm. It is outside of the scope of this text (and not in line with its objectives) to provide a more detailed description of the training process. The interested reader is referred to [39] for an in-depth mathematical description of the training process.

Loss function	Description
Mean squared error (MSE)	$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
Mean absolute error (MAE)	$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N}\lvert y_i - \hat{y}_i\rvert$
Binary cross-entropy	$L(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{N}\sum_{i=1}^{N}\Big(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\Big)$

Table 12: List of some of the most commonly used loss functions. Here $\mathbf{y} = (y_1, \ldots, y_N)$ denotes the vector of real values of the output variable, $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_N)$ denotes the vector of predicted values of the output variable, and N is the number of samples in the dataset. For further details on these and other loss functions, the reader is referred to [39, 87]

Implementation and analysis of a neural network

To illustrate the concepts introduced in this section, we present a simple application of a neural network to a prediction task. We train a neural network for the same use case as the fuzzy rule-based system presented in section 2.4, that is, the prediction of the crop yield (Y) of rice plant varieties based on the number of panicles per plant (P) and the number of grains per panicle (G). The dataset used in this example is the same as the one used for the implementation of the fuzzy rule-based system in section 2.4.

The neural network used here has two hidden layers with eight and four neurons each. The input layer has two neurons (corresponding to P and G, respectively) and the output layer has one neuron (corresponding to the predicted value for Y). Figure 9 shows a graphical representation of the neural network used in this example.

The activation function used in the hidden and output layers is the ReLU function. This activation function has been chosen for the hidden layers because it has been shown in the literature to perform better than other activation functions in such layers [87]; and it has been chosen for the output layer because the ReLU function returns a real number in $[0, +\infty)$, which is the desired range of the output variable Y.

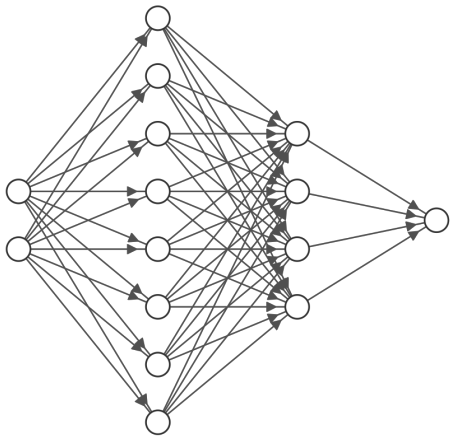

Figure 9: Graphical representation of the neural network used for the prediction of crop yields. Elaborated with *NN-SVG* [88].

In more formal terms, the neural network used in this example is defined by the following equations:

$$\mathbf{z}^{(1)}(\mathbf{x}) = \max\Big(\mathbf{0}, \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\Big), \tag{8}$$

$$\mathbf{z}^{(2)}(\mathbf{x}) = \max\Big(\mathbf{0}, \mathbf{W}^{(2)}\mathbf{z}^{(1)}(\mathbf{x}) + \mathbf{b}^{(2)}\Big), \tag{9}$$

$$y(\mathbf{x}) = \max\Big(\mathbf{0}, \mathbf{w}^{(3)}\mathbf{z}^{(2)}(\mathbf{x}) + b^{(3)}\Big), \tag{10}$$

where $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ is the input vector, $\mathbf{z}^{(1)}(\mathbf{x}) = (z_1^{(1)}(\mathbf{x}), \ldots, z_8^{(1)}(\mathbf{x})) \in \mathbb{R}^8$ is the vector of outputs of the neurons in the first hidden layer, $\mathbf{z}^{(2)}(\mathbf{x}) = (z_1^{(2)}(\mathbf{x}), \ldots, z_4^{(2)}(\mathbf{x})) \in \mathbb{R}^4$ is the vector of outputs of the neurons in the second hidden layer, and $y(\mathbf{x}) \in \mathbb{R}$ is the output of the neural network. The parameters of the neural network are the weights $\mathbf{W}^{(1)} \in \mathbb{R}^{8\times 2}$, $\mathbf{W}^{(2)} \in \mathbb{R}^{4\times 8}$ and $\mathbf{w}^{(3)} \in \mathbb{R}^4$, and the biases $\mathbf{b}^{(1)} \in \mathbb{R}^8$, $\mathbf{b}^{(2)} \in \mathbb{R}^4$ and $b^{(3)} \in \mathbb{R}$. Here the max function is applied element-wise to the input vectors.

In order to train and test the neural network, the dataset is split into a training set and a test set, with 80% and 20% of the samples, respectively. The training set is used to train the neural network, and the test set is used to test the performance of the neural network. The model is trained using the

mean squared error (MSE) as the loss function, over 100 iterations[18].

The neural network achieves an MSE of 0.3024 for the test set after 100 iterations. This corresponds to an RMSE (defined in (7)) for the predicted Y of 0.5499, which is significantly lower than the RMSE values obtained with the fuzzy rule-based system presented in section 2.4 (see Table 10). This result shows that the neural network is able to predict the crop yield with a higher degree of accuracy than the previous approach, which is also reflected in the plots of the predicted and the real crop yields for the test set, shown in Figure 10.

Figure 11 shows the value of the MSE for the training and test sets in each iteration. The plot shows that the MSE decreases significantly in the first iterations, and then decreases more slowly until it reaches a minimum. It is worth noting that the MSE for the test set is lower than the MSE for the training set in all the iterations. This behaviour is not common (the MSE for the test set is typically higher than the MSE for the training set). Although it may be an indicator of the good performance of the model, it may also be due to the small size of the dataset used in this example.

Last, it is worth extracting the weights of the connections between the different neurons in the neural network, in order to interpret the relationships between the input and output variables. Using the notation introduced in equations (8) to (10) the weights and biases of the neural network are the following:

$$\mathbf{W}^{(1)} = \begin{pmatrix} 0.1168 & -0.1180 \\ 0.1497 & -0.0343 \\ 0.6639 & 0.1744 \\ -0.1445 & -0.5547 \\ 0.0021 & -1.1560 \\ 0.1609 & 0.0969 \\ -0.2501 & 0.6321 \\ 0.0307 & 0.3202 \end{pmatrix}, \quad \mathbf{b}^{(1)} = \begin{pmatrix} 1.1781 \\ -0.4606 \\ -0.4792 \\ 1.4672 \\ 0.0298 \\ 0.6711 \\ 0.6602 \\ 1.3413 \end{pmatrix},$$

$$\mathbf{W}^{(2)} = \begin{pmatrix} -0.0336 & 0.0931 & -0.0170 & -0.1980 & -0.2004 & -0.1718 & -0.3212 & -0.2294 \\ 0.0610 & 0.2222 & 0.0028 & -0.1829 & 0.1443 & -0.1522 & -0.0875 & -0.2119 \\ 0.6226 & -0.2907 & 0.4525 & 0.7810 & -0.5792 & 0.5858 & 0.4467 & 0.6720 \\ 0.7013 & 0.0545 & 0.4847 & 0.8116 & -0.4004 & 0.5870 & 0.1678 & 0.5712 \end{pmatrix}, \quad \mathbf{b}^{(2)} = \begin{pmatrix} -0.0718 \\ 0.1838 \\ 0.8107 \\ 1.0152 \end{pmatrix},$$

$$\mathbf{w}^{(3)} = \begin{pmatrix} 0.2536 & -0.4154 & 0.8455 & 1.1169 \end{pmatrix}, \quad b^{(3)} = 1.0872.$$

In the case of feed-forward neural networks, the weights of the connections between one neuron and the next can be seen as a measure of the influence of that neuron to the next. By analyzing the chain of connections and their weights, we can extract information about the relationships between the input and output variables. For example, we observe that, among the neurons in the last hidden layer, the fourth one (with weight 1.1169) has the greatest influence on the output variable. Furthermore, we observe that all the neurons in the first hidden layer have a positive influence on the fourth neuron in the last hidden layer (evidenced by the positive values in the last row of $\mathbf{W}^{(2)}$).

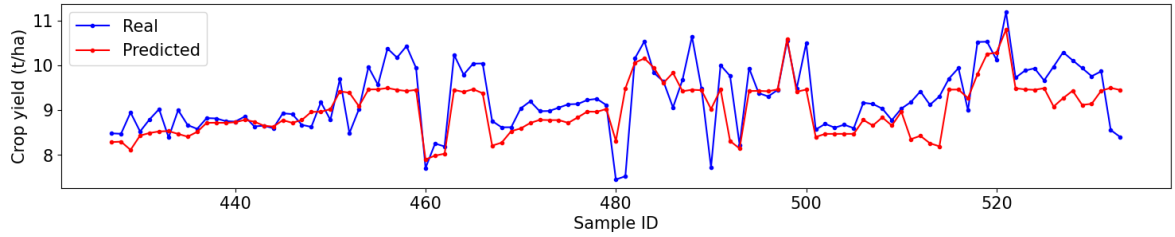

Figure 10: Real and predicted crop yields for the samples in the test set, using the neural network.

[18]For the reader interested in technical details about the training process, the neural network implemented here has been trained using the Adam algorithm (a variant of the gradient descent algorithm) [89], with a learning rate of 0.001, for the optimization of the parameters. This optimizer and learning rate have shown the best performance in preliminary tests involving many different trials with different optimization algorithms and learning rates. It is worth noting that the input of the network has been scaled appropiately to have zero mean and unit variance, in order to improve the performance of the training process.

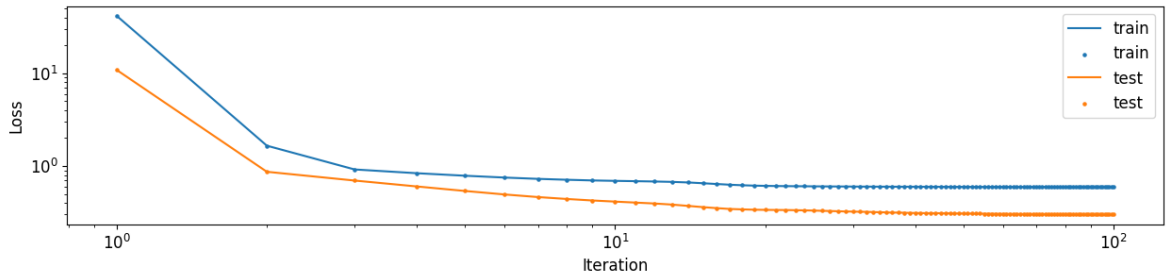

Figure 11: MSE loss for the training and test sets in every iteration of the training process for the neural network. Both axes are in logarithmic scale.

However, it becomes apparent that extracting this information is not straightforward, and it becomes less clear as the number of neurons and layers in the neural network increases[19]. This exemplifies how the complexity inherent in neural networks makes it difficult to extract information about the relationships between the input and output variables, which leads to a lack of interpretability. Furthermore, the use of activation functions also makes the operations of the neurons less interpretable.

Therefore, although the use of neural networks for the prediction of crop yields gives better results in terms of accuracy, it is not as interpretable as the fuzzy rule-based system presented in section 2.4. In subsequent sections, we will see an approach that aims to combine the accuracy of neural networks with the interpretability of fuzzy logic, and which shows positive results in this regard.

The neural network presented in this example has been implemented and trained using the PyTorch library [90] in the Python 3 programming language. The code for the implementation, training and testing of the neural network is available in the Github repository found online at: https://github.com/loredanasandu/tfg-fuzzy-logic-artificial-intelligence. An extract with the most relevant parts of the code can be found in Appendix A, section A.3.

3.2 Fuzzy neural networks

Fuzzy neural networks [31] are an extension of classical neural networks in which the classical neurons are replaced by *fuzzy neurons*. Recall that a non-fuzzy neuron is a function of the form

$$y(\mathbf{x}) = a(\mathbf{x} \cdot \mathbf{w} + b) = a\Big(\sum_{i=1}^{n} w_i \cdot x_i + b\Big)$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ is the input vector, $\mathbf{w} = (w_1, \ldots, w_n)$ is the weight vector, b is the bias, and a is the activation function. For the sake of simplicity in the argument that follows, we consider $b = 0$[20].

In such neuron, two computations are performed: (i) the products of the inputs x_i and the weights w_i, and (ii) the summation of these products.

We present two types of fuzzy neurons [91]. On the one hand, *fuzzy or-neurons* are constructed by replacing the products by *and* ($\wedge$) operations, modelled with a t-norm, and replacing the summation by an *or* ($\vee$) operation, modelled as a t-conorm. The resulting fuzzy neuron is a function of the form

$$y(\mathbf{x}) = \bigvee_{i=1}^{n} (x_i \wedge w_i) = \mathop{\mathbf{S}}_{i=1}^{n} (x_i \,\mathbf{t}\, w_i)$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ is the input vector, $\mathbf{w} = (w_1, \ldots, w_n)$ is the weight vector, and $\vee$ and $\wedge$ are

[19] It is worth noting that the neural network presented here is intended to be simple and with low dimensionality in order to illustrate the concepts and facilitate the interpretation of the results. The models commonly used in literature and the industry are much more complex, with many more neurons and layers.

[20] In the argument that follows, we define neurons which are based on fuzzy logic operators, with the aim of extracting logical relationships between the variables. Removing the bias simplifies the model and, most importantly for our purposes, it simplifies its interpretation. The addition of the bias to the model and the analysis of how it affects the results is left for future work.

modelled by a t-conorm $\mathbf{S}$ and a t-norm $\mathbf{t}$, respectively.

On the other hand, *fuzzy and-neurons* are constructed by replacing the products by $\vee$ operations, and replacing the summation by an $\wedge$ operation. The resulting fuzzy neuron is a function of the form

$$y(\mathbf{x}) = \bigwedge_{i=1}^{n} (x_i \vee w_i) = \underset{i=1}{\overset{n}{\mathbf{T}}} (x_i \,\mathbf{s}\, w_i)$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ is the input vector, $\mathbf{w} = (w_1, \ldots, w_n)$ is the weight vector, and $\wedge$ and $\vee$ are modelled by a t-norm $\mathbf{T}$ and a t-conorm $\mathbf{s}$, respectively.

Recall that the activation function a included in non-fuzzy neurons is typically a non-linear function that makes the neural network approximate non-linear functions. Therefore, by using non-linear t-norms and t-conorms, the activation function a is not necessary in fuzzy neurons.

A fuzzy neural network is a composition of fuzzy neurons, analogously to the way that a classical neural network is a composition of classical neurons, described in section 3.1. The input of the fuzzy neural network is considered in terms of membership functions associated to designated fuzzy sets (that is, the crisp input is fuzzified, as described in section 2.4). Moreover, both the original membership values of the input, $\mu(x_i)$, and the corresponding negations, $1 - \mu(x_i)$ are considered (thus making the input of dimenstion $2n$), in order to account for both positive and negative contributions of the input in its relationship to the output[21].

We consider each hidden layer to be composed either entirely of fuzzy and-neurons (*and-layer*), or entirely of fuzzy or-neurons (*or-layer*)[22]. The output layer is composed entirely of or-neurons, with each neuron corresponding to an output fuzzy set. We use or-neurons in this layer with the aim of aggregating the multiple membership values obtained for the same fuzzy set, as described in section 2.4. The fuzzy output is then defuzzified in order to obtain crisp predicted values.

Figure 12 shows the scheme for an implementation of a hypothetical fuzzy neural network with one hidden and-layer and one hidden or-layer. Note how the process for implementing this system integrates elements from fuzzy rule-based systems and from classical neural networks.

Interpretation of fuzzy neurons in terms of logical relationships

The operations performed by fuzzy neurons can be interpreted as a realization of logical relationships. We now describe how this interpretation is done. Consider the same notation introduced above, with $\mathbf{x} = (x_1, \ldots, x_n)$ is the input vector, $\mathbf{w} = (w_1, \ldots, w_n)$ is the weight vector.

Or-neurons combine the input values x_i with the weights w_i via the $\wedge$ operation, and aggregate the result with the $\vee$ operation. This is a realization of the logical disjunction

$$\alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_n$$

with α_i denoting the proposition $x_i \wedge w_i$. This is interpreted as a disjunction between the x_i's, where the influence of each x_i is determined (weighted) by the corresponding w_i [31]. Due to the nature of the $\wedge$ operation, higher values of w_i give a stronger influence of x_i in the output of the neuron[23].

[21] It may be argued that considering the corresponding negations of each membership value contributes to role that a high-dimensionality of the input plays in the lack of interpretability of a neural network. However, the fuzzification of the input allows us to interpret the input in terms of fuzzy sets, which is a more interpretable approach than the use of raw data. Also, the use of the corresponding negations allows us to derive more complex logical relationships between the input and output variables. Furthermore, the fuzzification of the input may be designed in order to reduce the dimensionality of the input. For instance, in computer vision or image recognition, instead of using images as inputs to the model (high-dimensional inputs) one can define fuzzy sets that represent specific characteristics useful for the problem at hand, such as colour [92].

[22] In practical terms, this does not signify a significant limitation or loss of generality. It is common in the literature and applications to consider each hidden layer to be composed entirely of fuzzy and-neurons or entirely of fuzzy or-neurons. This facilitates the implementation of the model and the interpretation of the results.

[23] Note that the weights w_i can therefore be interpreted as a measure of the influence of x_i in the logical relationship. This interpretation is in accordance with the one given for the weights in classical neural networks (section 3.1).

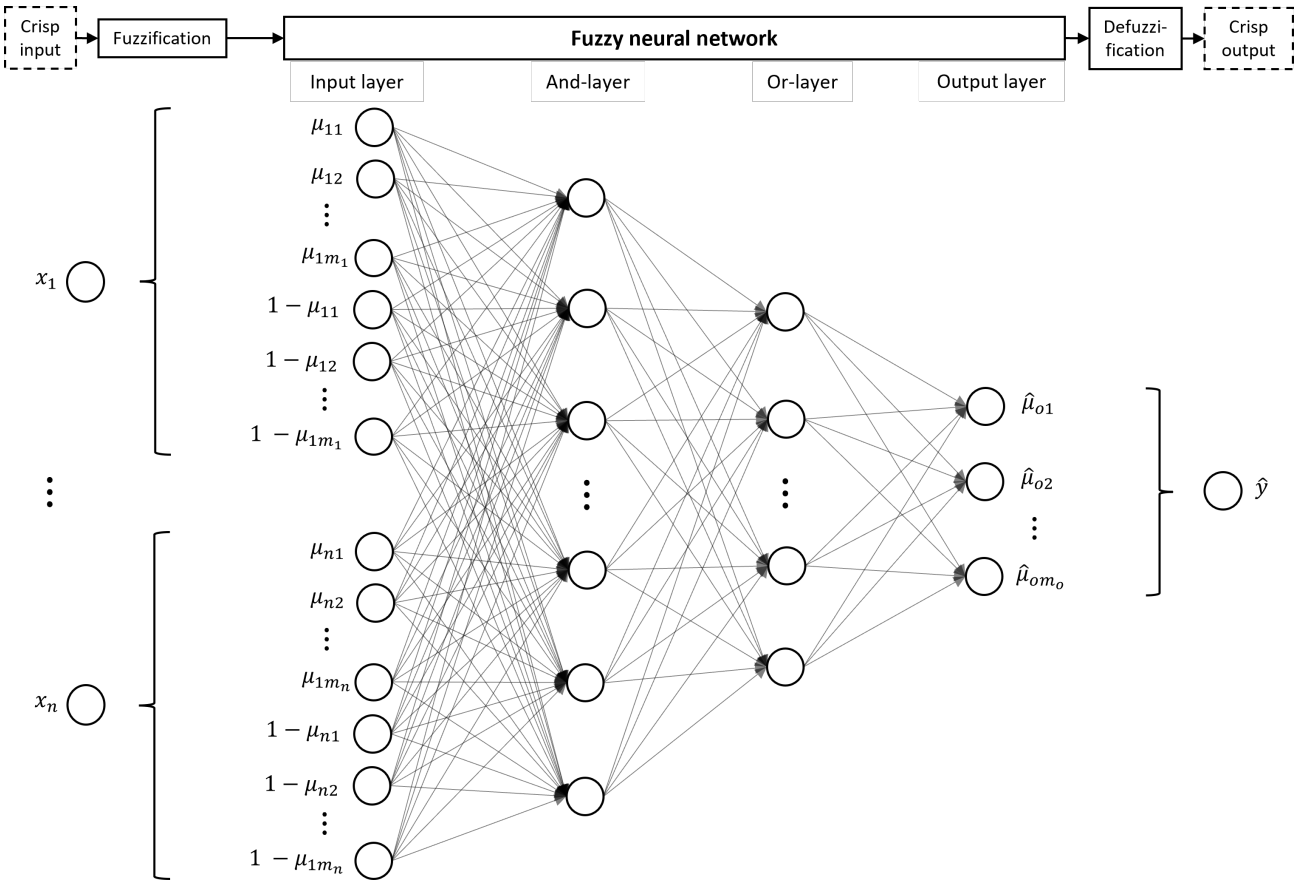

Figure 12: Scheme for the implementation of a hypothetical fuzzy neural network with one hidden and-layer and one hidden or-layer. In this representation, $x_1, \dots x_n$ are the crisp input variables to the system, and we fuzzify each x_i to have m_i fuzzy sets, with corresponding membership functions $\mu_{i1}, \dots, \mu_{im_i}$. We also consider the negations $1-\mu_{i1}, \dots, 1-\mu_{im_i}$, $i \in \{1, \dots, n\}$. The fuzzy neural network is composed of the input layer (which is given the fuzzified input), a hidden and-layer, a hidden or-layer, and the output layer, which is composed of or-neurons. The fuzzy neural network outputs the fuzzified predicted values, in terms of the membership values $\hat{\mu}_{o1}, \dots, \hat{\mu}_{om_o}$. Last, the output is defuzzified to obtain the crisp predicted value $\hat{y}$.

In the case of and-neurons, the input values x_i are combined with the weights w_i via the $\vee$ operation, and the result is aggregated with the $\wedge$ operation. This is a realization of the logical conjunction

$$\beta_1 \wedge \beta_2 \wedge \cdots \wedge \beta_n.$$

with β_i denoting the proposition $x_i \vee w_i$. This is interpreted as a conjunction between the x_i's, where, similarly to the case of or-neurons, the influence of each x_i is also determined by the corresponding w_i. However, the opposite weighting effect takes place: due to the nature of the $\vee$ operation, higher values of w_i give a weaker influence of x_i in the output of the neuron.

Since a fuzzy neural network is a composition of fuzzy neurons, it can be interpreted as a (weighted) composition of logical operations between the input membership values. This yields logical rules with which to infer the output membership values from the input membership values. The process of interpreting and extracting logical rules from a fuzzy neural network will be illustrated in the following subsection.

Implementation and analysis of a fuzzy neural network

We illustrate the implementation and analysis of a fuzzy neural network considering the prediction of crop yields of rice plants. We will compare this approach with the systems presented in section 2.4 and section 3.1.

Recall that our objective is to predict the crop yield (Y) of rice plant varieties based on the number

of panicles per plant (P) and the growth period of the plant (G). The dataset used in this example is the same as the one used for the implementations in section 2.4 and section 3.1. We also define the same fuzzy sets for the input and output variables as in section 2.4 (see Table 4 for the notation and Table 5 for the membership functions of the fuzzy sets).

The fuzzy neural network used here has two hidden layers: one and-layer with 20 neurons and one or-layer with 10 neurons. The input layer has 20 neurons (corresponding to the fuzzified inputs $(\mu_{VLP}, \ldots, \mu_{VHP}, \mu_{VLG}, \ldots, \mu_{VHG})$ and their negations $(1-\mu_{VLP}, \ldots, 1-\mu_{VHP}, 1-\mu_{VLG}, \ldots, 1-\mu_{VHG})$), and the output layer has 5 neurons (corresponding to the predicted membership values for the output fuzzy sets $(p_{VLY}, \ldots, p_{VHY})$). For simplicity, we denote the input vector of membership values for P and G as $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_{20})$ and the output vector of predicted membership values for Y as $\hat{\boldsymbol{\mu}} = (\hat{\mu}_1, \ldots, \hat{\mu}_5)$.

In formal terms, the fuzzy neural network used in this example is defined by the following equations:

$$z_m^{(1)}(\boldsymbol{\mu}) = \mathop{\mathbf{T}}_{i=1}^{20} \Big(\mu_i \ \mathbf{s} \ w_{i,m}^{(1)} \Big), \tag{11}$$

$$z_n^{(2)}(\boldsymbol{\mu}) = \mathop{\mathbf{S}}_{i=1}^{20} \Big(z_i^{(1)}(\boldsymbol{\mu}) \ \mathbf{t} \ w_{i,n}^{(2)} \Big), \tag{12}$$

$$\hat{\mu}_k(\boldsymbol{\mu}) = \mathop{\mathbf{S}}_{i=1}^{10} \Big(z_i^{(2)}(\boldsymbol{\mu}) \ \mathbf{t} \ w_{i,k}^{(3)} \Big), \tag{13}$$

where $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_{20})$ denotes the vector of input membership values, $z_m^{(1)}(\boldsymbol{\mu})$ is the output of the m-th neuron in the hidden and-layer ($m \in \{1, \ldots, 20\}$), $z_n^{(2)}(\boldsymbol{\mu})$ is the output of the n-th neuron in the hidden or-layer ($n \in \{1, \ldots, 10\}$), and $\hat{\mu}_k(\boldsymbol{\mu})$ is the predicted membership value for the k-th output fuzzy set, with $k \in \{1, \ldots, 5\}$. The parameters of the fuzzy neural network are the weights $\mathbf{W}^{(1)} = \left(w_{i,m}^{(1)}\right)_{i,m=1,1}^{20,20}$, $\mathbf{W}^{(2)} = \left(w_{i,n}^{(2)}\right)_{i,n=1,1}^{20,10}$ and $\mathbf{W}^{(3)} = \left(w_{i,k}^{(3)}\right)_{i,k=1,1}^{10,5}$.

The defuzzification of the predicted membership values is performed in the same way as described in section 2.4, that is, using the centroid method. This gives the predicted crisp value for the output variable Y.

In order to train and test the system, the dataset is split into a training set and a test set, with 80% and 20% of the samples, respectively, as it was done in section 3.1. The model is trained using MSE as the loss function[24], over 25 iterations[25].

The model has been trained with all the dual pairs (Table 1). It gives the best results when using the product t-norm and the sum t-conorm, $\langle t_\pi, s_\pi \rangle$[26]. For the prediction of membership values, the model achieves an MSE of 0.0898 in the test set after 25 iterations. After the defuzzification, the system achieves an RMSE (defined in (7)) of 0.53715 for the prediction of Y, in the test set. Figure 13 shows the value of the MSE for the predicted membership values in the training and test sets in

[24] Here the model is trained to minimize the MSE between the predicted membership values (which are the output of the model) and the real membership values for the fuzzy sets for Y, given each sample in the dataset. More specifically, the loss function is defined as

$$\mathrm{MSE}(\boldsymbol{\mu}_{\boldsymbol{Y}}, \hat{\boldsymbol{\mu}}_{\boldsymbol{Y}}) = \frac{1}{5} \sum_{i=1}^{5} (\mu_i - \hat{\mu}_i)^2,$$

where $\boldsymbol{\mu}_{\boldsymbol{Y}} = (\mu_1, \ldots, \mu_5)$ denotes the vector of real membership values for the output fuzzy sets VLY, LY, MY, HY and VHY for one sample of the dataset, and $\hat{\boldsymbol{\mu}}_{\boldsymbol{Y}} = (\hat{\mu}_1, \ldots, \hat{\mu}_5)$ denotes the vector of predicted membership values for the output fuzzy sets given the same sample.

[25] Regarding technical details about the training process, the fuzzy neural network implemented here has been trained using the Adam algorithm [89], with a learning rate of 0.1, which has shown the best performance in preliminary tests.

[26] When using the dual pair $\langle t_{\min}, s_{\max} \rangle$, the model learns (that is, optimizes its parameters) exceedingly slowly. More specifically, the loss function decreases very slowly in the first iterations, and then the decrease stops (or it is negligible) in the following iterations. Regarding technical considerations, these results have been obtained using the Adam algorithm the optimization of the parameters, with a learning rate of 0.0001. It is the low learning rate that leads to the slow pace at which the model learns in these conditions. However, the model does not learn at all when using a higher learning rate. The Stochastic Gradient Descent (SGD) algorithm [93] for the optimization of the parameters has also been tested. With this algorithm, the loss function stays constant regardless of the learning rate used.

When using the dual pair $\langle t_{\mathrm{L}}, s_{\mathrm{L}} \rangle$, the model is unable to learn, which is evidenced by the fact that the loss function does not decrease (nor increase) in any of the iterations. The same results have been obtained for with both the Adam and the SGD algorithm, for learning rates of different orders of magnitude.

More details and discussion about these results, as well as possible reasons, will be presented in section 4.

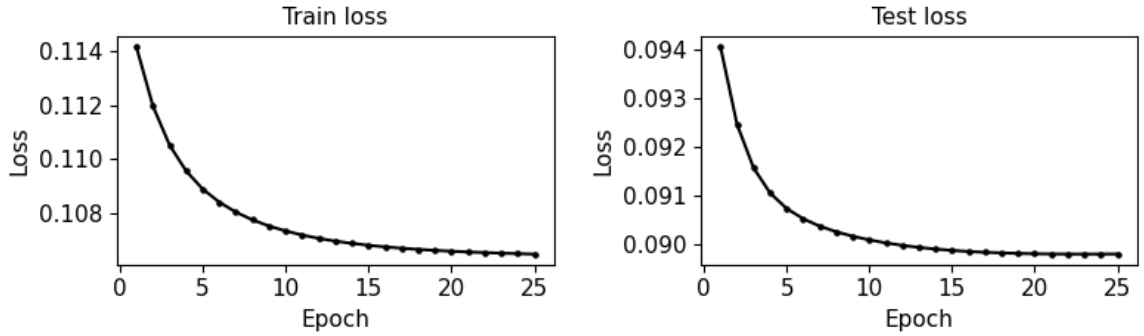

Figure 13: Loss for the predicted membership values in the training set (left) and test set (right) in every iteration of the training process of the fuzzy neural network, in terms of the MSE.

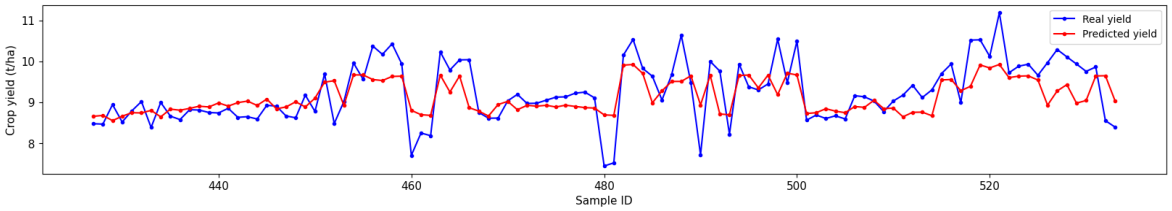

Figure 14: Real and predicted crop yields by the fuzzy neural network for the samples in the test set.

each iteration. Figure 14 shows the real and the predicted crisp values for Y in the test set.

The RMSE of 0.53715 obtained with the fuzzy neural network is significantly lower than the RMSE of 0.8606 obtained with the fuzzy rule-based system presented in section 2.4, using the dual pair $\langle t_\pi, s_\pi \rangle$. This shows that the fuzzy neural network is able to predict the crop yield with a higher degree of accuracy than the fuzzy rule-based system. This result is expected, since the fuzzy rule-based system evaluates fixed rules, whereas the fuzzy neural network is able to approximate more complex relationships between the input and output variables and optimize the operations accordingly.

We also observe that the RMSE obtained with the fuzzy neural network is similar to the RMSE of 0.5499 obtained with the neural network presented in section 3.1. This shows that the ability of the fuzzy neural network to approximate the output variable is comparable to that of the neural network for the use case at hand. Furthermore, the fuzzy neural network is able to achieve this result while maintaining a high degree of interpretability, as we will see in the following paragraphs.

By extracting the weights of the connections between the neurons in the fuzzy neural network, we can interpret the relationships between the input and output variables in terms of fuzzy logic rules. This is done by analyzing the chain of operations performed (which model logical operations) and their corresponding weights (which model the influence of each neuron on the next). Figure 15 shows the weights of the fuzzy neural network.

Recall that, for or-neurons, the greater the weight associated to an input, the greater the influence of that input in the neuron (due to the $\wedge$ operation between the input and the weight). Conversely, in the case of and-neurons, the lower the weight associated to an input, the greater the influence of that input in the neuron (due to the $\vee$ operation between the input and the weight). Therefore, in order to extract the rules that have the greatest influence in the output of the system, we apply a threshold to the weights in each layer. We select weigths lower than 0.1 in the first layer (matrix $\mathbf{W}^{(1)}$), weights greater than 0.9 in the second layer (matrix $\mathbf{W}^{(2)}$), and weights greater than 10.0 in the third layer (matrix $\mathbf{W}^{(3)}$).

It is important to note here that the weights in the third layer have been automatically optimized in the training process to be outside the desired range of $[0, 1]$. After performing preliminary explorations of this phenomenon, we believe that this inconvenience may be caused by the algorithm used for the

$$\mathbf{W}^{(1)} = \begin{pmatrix}
0.61 & 0.38 & 0.64 & 0.47 & 0.71 & 0.62 & 0.44 & 0.1 & 0.61 & \mathbf{0.06} & 0.57 & 0.53 & 0.39 & 0.91 & 0.53 & 0.71 & 0.71 & 0.21 & 0.31 & 0.98 \\
\mathbf{0.01} & 0.47 & 0.46 & 0.85 & 0.45 & 0.63 & 0.48 & 0.22 & 0.22 & 0.26 & \mathbf{0.05} & 0.18 & 0.62 & 0.83 & 0.52 & 0.27 & 0.72 & 0.31 & 0.39 & 0.23 \\
0.34 & \mathbf{0.04} & 0.71 & 0.69 & 0.6 & 0.75 & 0.71 & 0.52 & 0.55 & 0.54 & 0.77 & 0.84 & 0.86 & 0.79 & 0.38 & 0.48 & 0.4 & 0.79 & 0.56 & 0.96 \\
0.75 & \mathbf{0.07} & 0.65 & 0.98 & 0.94 & 0.49 & 0.67 & \mathbf{0.03} & 0.34 & 0.74 & \mathbf{0.04} & 0.94 & 0.17 & 0.66 & 0.48 & 0.59 & 0.55 & \mathbf{0.03} & 0.39 & 0.18 \\
0.93 & 0.44 & \mathbf{0.0} & 0.62 & 0.72 & 0.28 & 0.45 & 0.72 & 0.19 & 0.24 & 0.45 & 0.15 & 0.81 & 0.54 & 0.8 & 0.77 & 0.11 & 0.19 & 0.16 & 0.34 \\
0.32 & 0.42 & \mathbf{0.02} & 0.71 & 0.78 & 0.66 & 0.82 & 0.88 & \mathbf{0.01} & 0.58 & 0.96 & 0.14 & 0.28 & 0.47 & 0.19 & 0.41 & 0.43 & 0.95 & 0.47 & 0.77 \\
0.87 & 0.72 & \mathbf{0.07} & 0.87 & 0.83 & 0.14 & 0.32 & 0.84 & \mathbf{0.01} & \mathbf{0.06} & 0.16 & 0.66 & 0.3 & \mathbf{0.05} & 0.69 & 0.75 & 0.69 & \mathbf{0.07} & 0.99 & 0.46 \\
\mathbf{0.02} & 0.42 & 0.83 & 0.73 & 0.49 & 0.85 & \mathbf{0.04} & 0.85 & 1.0 & 0.2 & 0.61 & 0.42 & \mathbf{0.08} & 0.5 & 0.33 & \mathbf{0.07} & 0.14 & 0.51 & 0.96 & 0.68 \\
0.17 & 0.84 & 0.54 & \mathbf{0.01} & 0.52 & 0.82 & 0.21 & 0.48 & 0.25 & 0.11 & 0.22 & 0.55 & 0.82 & 0.41 & \mathbf{0.05} & 0.5 & \mathbf{0.07} & 0.53 & 0.87 & 0.71 \\
0.19 & 0.39 & 0.7 & 0.64 & \mathbf{0.03} & 0.21 & 0.32 & 0.68 & 0.95 & 0.72 & 0.69 & \mathbf{0.08} & 0.82 & 0.68 & 0.34 & 0.26 & 0.64 & 0.23 & 0.7 & 0.88 \\
0.17 & 0.19 & 0.29 & \mathbf{0.07} & 0.44 & 0.81 & 0.69 & 0.34 & 0.44 & 0.28 & 0.1 & 0.36 & 0.53 & 0.25 & 0.13 & 0.65 & 0.57 & 0.34 & 0.64 & 0.96 \\
0.53 & 0.16 & 0.96 & 0.86 & 0.45 & 0.27 & 0.29 & 0.94 & 0.76 & 0.24 & 0.52 & 0.6 & 0.63 & 0.47 & 0.24 & 0.48 & 0.36 & 0.33 & 0.25 & 0.26 \\
0.26 & 0.7 & \mathbf{0.08} & \mathbf{0.02} & 0.93 & 0.68 & 0.78 & 0.72 & 0.31 & 0.19 & 0.16 & 0.47 & 0.62 & \mathbf{0.03} & 0.12 & 0.78 & 0.56 & 0.83 & 0.95 & 0.8 \\
0.39 & 0.28 & 0.64 & 0.91 & \mathbf{0.02} & 0.63 & 0.11 & 0.1 & 0.68 & 0.75 & 0.67 & 0.65 & 0.35 & 0.51 & \mathbf{0.06} & 0.17 & 0.51 & 0.39 & 0.42 & 0.78 \\
0.69 & 0.39 & 0.34 & 0.14 & 0.76 & 0.15 & 0.66 & 0.41 & 0.89 & 0.18 & 0.17 & 0.56 & 0.51 & 0.28 & 1.0 & \mathbf{0.07} & 0.77 & 0.74 & 0.38 & 0.68 \\
0.42 & 0.12 & 0.83 & 0.37 & 0.43 & 0.99 & 0.98 & 0.13 & 0.37 & 0.78 & 0.81 & 0.13 & 0.14 & 0.71 & 0.16 & 0.86 & 0.12 & 0.21 & 0.69 & 0.77 \\
0.32 & 0.21 & 0.69 & 0.66 & 0.63 & \mathbf{0.04} & 0.35 & 0.32 & 0.51 & 0.95 & 0.14 & 0.49 & 0.69 & 0.78 & \mathbf{0.0} & 0.52 & 0.89 & 0.17 & 0.81 & 0.43 \\
0.95 & \mathbf{0.07} & 0.59 & 0.86 & 0.86 & 0.25 & \mathbf{0.09} & 0.51 & 0.26 & 0.82 & 0.24 & 0.55 & \mathbf{0.06} & 0.5 & 0.94 & 0.77 & 0.71 & 0.21 & 0.93 & 0.25 \\
0.96 & \mathbf{0.09} & 0.25 & 0.35 & 0.58 & 0.77 & 0.61 & 0.31 & 0.1 & 0.35 & 0.26 & 0.2 & 0.81 & 0.57 & 0.23 & 0.74 & 0.16 & 0.21 & 0.45 & 0.83 \\
0.76 & 0.16 & 0.15 & 0.67 & 0.46 & 0.55 & 0.51 & 0.14 & 0.42 & 0.34 & 0.41 & 0.81 & \mathbf{0.03} & 0.55 & 0.99 & \mathbf{0.05} & 0.68 & 1.0 & 0.78 & 0.77
\end{pmatrix}$$

$$\mathbf{W}^{(2)} = \begin{pmatrix}
0.31 & 0.71 & 0.56 & \mathbf{0.98} & 0.02 & 0.76 & 0.19 & 0.05 & \mathbf{0.97} & 0.6 \\
0.22 & 0.03 & 0.4 & 0.39 & 0.46 & 0.35 & 0.64 & 0.84 & 0.1 & 0.31 \\
\mathbf{0.93} & 0.02 & 0.9 & 0.4 & 0.64 & 0.24 & 0.74 & 0.56 & 0.38 & \mathbf{0.97} \\
0.05 & \mathbf{0.93} & 0.84 & 0.42 & 0.34 & 0.9 & 0.7 & 0.06 & 0.11 & 0.01 \\
0.29 & 0.42 & 0.68 & 0.79 & 0.05 & 0.76 & 0.67 & 0.66 & 0.04 & 0.17 \\
0.44 & 0.74 & 0.38 & 0.34 & 0.55 & 0.54 & 0.44 & 0.54 & 0.24 & \mathbf{0.92} \\
0.53 & 0.75 & 0.08 & 0.35 & 0.67 & 0.17 & 0.52 & 0.23 & 0.34 & 0.52 \\
0.11 & 0.66 & 0.82 & 0.06 & 0.47 & 0.39 & 0.05 & 0.39 & \mathbf{0.91} & 0.56 \\
0.48 & \mathbf{0.96} & 0.65 & 0.89 & 0.1 & 0.77 & 0.89 & 0.73 & 0.43 & \mathbf{0.95} \\
0.25 & 0.68 & 0.0 & 0.13 & 0.04 & 0.11 & 0.3 & 0.06 & 0.16 & 0.24 \\
0.73 & \mathbf{0.97} & 0.76 & 0.28 & 0.21 & 0.76 & 0.2 & 0.09 & 0.3 & \mathbf{0.97} \\
\mathbf{0.91} & 0.08 & \mathbf{0.99} & 0.23 & 0.28 & 0.04 & \mathbf{0.92} & 0.89 & \mathbf{0.97} & 0.65 \\
0.3 & 0.79 & 0.37 & 0.03 & 0.23 & 0.79 & 0.05 & 0.0 & \mathbf{0.95} & 0.09 \\
0.61 & \mathbf{0.98} & 0.63 & 0.44 & 0.68 & \mathbf{0.98} & 0.14 & 0.74 & 0.3 & 0.18 \\
0.47 & 0.81 & 0.09 & 0.38 & 0.37 & 0.78 & 0.69 & 0.75 & 0.89 & 0.77 \\
0.3 & 0.56 & 0.88 & 0.81 & 0.15 & 0.5 & 0.71 & 0.67 & 0.49 & 0.46 \\
0.7 & 0.29 & 0.82 & 0.12 & 0.07 & 0.56 & 0.59 & 0.66 & 0.18 & 0.64 \\
0.08 & 0.74 & 0.82 & 0.26 & 0.59 & 0.82 & 0.42 & 0.69 & 0.18 & 0.63 \\
0.65 & 0.59 & 0.17 & 0.52 & 0.82 & 0.12 & 0.64 & 0.28 & 0.26 & 0.17 \\
0.74 & 0.86 & 0.08 & 0.89 & 0.79 & 0.61 & 0.85 & 0.61 & 0.58 & 0.83
\end{pmatrix}, \quad \mathbf{W}^{(3)} = \begin{pmatrix}
6.34 & 7.93 & 5.37 & -6.02 & 6.1 \\
1.97 & 1.47 & 6.51 & 4.04 & -15.31 \\
-17.55 & -4.2 & 5.17 & \mathbf{18.99} & \mathbf{10.48} \\
1.69 & -1.47 & -3.64 & 1.34 & \mathbf{18.55} \\
\mathbf{11.53} & 7.75 & \mathbf{12.04} & -4.57 & -22.35 \\
-8.51 & -4.54 & -1.01 & 1.39 & -4.27 \\
0.53 & -6.74 & -4.33 & 4.87 & \mathbf{13.96} \\
-1.41 & 3.65 & -0.7 & -4.96 & \mathbf{29.07} \\
3.84 & \mathbf{20.16} & \mathbf{23.76} & \mathbf{15.06} & -14.72 \\
\mathbf{10.86} & 5.05 & 1.75 & -5.84 & -1.72
\end{pmatrix}$$

Figure 15: Weights of the connections between the neurons in the fuzzy neural network. The values in bold are the ones selected after applying the threshold conditions. The notation used is the same as in equations (11) to (13). The values are rounded to two decimal places.

optimization of the parameters, which is incompatible with constraining the values for the weights[27]. A more in-depth analysis would be needed to determine exact causes and solutions, which would involve a more detailed study of the optimization algorithm and the training process. This is outside of the scope and objectives of this text, and is proposed as future work.

Furthermore, it is also worth noting that the weights in the last layer are the ones with the greatest variation in the range of values. Preliminary tests have shown that the last layer is the only one that has been optimized significantly in the training process. The change of the weights in the previous layers has been negligible; these weights have remained very close to their initial values. This phenomenon may be due to the fact that the model has been trained with a small dataset.

By analyzing the chain of operations performed by the model using the selected weights, we can extract the rules that have the greatest influence in the output of the system. As an example, consider the output neuron $\hat{\mu}_1$, corresponding to the membership function μ_{VLY}. The neurons which have the most influence on $\hat{\mu}_1$, according to the threshold imposed, are $z_5^{(2)}$ and $z_{10}^{(2)}$, resulting in the following operation for $\hat{\mu}_1$:

$$\hat{\mu}_1 = z_5^{(2)} \vee z_{10}^{(2)}.$$

[27]Gradient-based optimization algorithms (such as the Adam algorithm used here) are typically used for the training of neural networks and have been used here as well. These algorithms for automatic optimization are not compatible with constraining the values for the weights. However, there are other optimization algorithms that are compatible with this constraint. Further discussion and suggestions will be provided in section 4.

According to the selection of the weights in the second layer, the neuron $z_5^{(2)}$ is not significantly influenced by any of the neurons before it, but we obtain the following operation for $z_{10}^{(2)}$:

$$z_{10}^{(2)} = z_3^{(1)} \vee z_6^{(1)} \vee z_9^{(1)} \vee z_{11}^{(1)}.$$

In turn, the values for these neurons are computed with the following relations:

$$z_3^{(1)} = \mu_5 \wedge \mu_6 \wedge \mu_7 \wedge \mu_{13}, \qquad z_6^{(1)} = \mu_{17}, \qquad z_9^{(1)} = \mu_6 \wedge \mu_7, \qquad z_{11}^{(1)} = \mu_2 \wedge \mu_4.$$

Putting it all together and simplyfying the expressions where possible, we obtain the following relation between $\hat{\mu}_1$ and the input neurons:

$$\begin{aligned}\hat{\mu}_1 &= (\mu_5 \wedge \mu_6 \wedge \mu_7 \wedge \mu_{13}) \vee (\mu_{17}) \vee (\mu_6 \wedge \mu_7) \vee (\mu_2 \wedge \mu_4) \\ &= (\mu_6 \wedge \mu_7) \vee (\mu_{17}) \vee (\mu_2 \wedge \mu_4) = (\mu_{VLG} \wedge \mu_{LG}) \vee (\neg\mu_{LG}) \vee (\mu_{LP} \wedge \mu_{HP}),\end{aligned}$$

which corresponds to the following rule:

(R1) IF (P is LP AND P is HP) OR (G is VLG AND G is LG) OR (G is NOT LG) THEN (Y is VLY).

We extract the rules for the rest of output neurons in the same way. The rest of the extracted rules are the following:

(R2) IF (P is LP AND G is MG) OR (P is HP) OR (G is VHG) OR (G is MG AND G is NOT MG AND G is NOT VHG) THEN (Y is LY),

(R3) IF (P is LP AND G is MG) OR (P is HP) OR (G is VHG) OR (G is MG AND G is NOT MG AND G is NOT VHG) THEN (Y is MY),

(R4) IF (P is LP AND G is MG) OR (P is HP) OR (G is VHG) OR (G is MG AND G is NOT MG AND G is NOT VHG) THEN (Y is HY),

(R5) IF (G is VHG) OR (P is LP AND G is MG) THEN (Y is VHY).

We observe that rules (R2)-(R4) share the same antecedent, and have different consequents. Also, we find expressions which may appear conflicting in terms of Boolean logic (such as the expression "G is MG AND G is NOT MG" in rules (R2)-(R4)). These may appear contradictory in the context of classical logic. However, in the context of fuzzy logic, the fuzzy sets for each variable are not disjoint. Therefore, these expressions are valid in the context of fuzzy logic.

It is important to note that the rules obtained are optimized in the specific context of the problem and with the data used here. Therefore, we need to remain cautious when generalizing these rules to other contexts or other data. Furthermore, the model is proposed here as an example to illustrate the implementation and interpretation of fuzzy neural networks. With additional data and better training, the model may achieve better results and yield more meaningful rules, that can be more adequate for generalization to other data in the same context.

Overall, we have shown that fuzzy neural networks offer an enhanced interpretability, while still maintaining an accuracy in the predicition comparable to that of classical neural networks. The RMSE obtained with the fuzzy neural network is lower than the one obtained with the neural network (section 3.1) and significantly lower than the one obtained with the fuzzy rule-based system (section 2.4). Table 13 summarizes the RMSE obtained with the three models[28].

The most notable advantage of the model presented here is the ability of optimizing rules that can used to explain the process used to infer the output variable. The model is able to extract rules that are meaningful in the context of the problem at hand, and that can be used to explain the relationships between the input and output variables. This is a significant advantage over other machine learning models, such as neural networks, which are not able to provide this level of interpretability. Further-

[28]Note that the RMSE is a measure of the error of the result in the prediction task. Therefore, it is used as a measure of the accuracy of the model in the sense that, the lower the RMSE, the higher the accuracy of the model.

more, this interpretability does not necessarily come at the expense of accuracy in the prediction task. The model is able to generate interpretable results, while still maintaining an adequate accuracy in the predicion, comparable to that of the classical approaches.

The fuzzy neural network presented here has been implemented and trained using the PyTorch library [90] in the Python 3 programming language. The code for the implementation, training and testing of the fuzzy neural network is available in the Github repository found online at: https://github.com/loredanasandu/tfg-fuzzy-logic-artificial-intelligence. An extract with the most relevant parts of the code is included in Appendix A, section A.4.

Model	**Dual pair**	**RMSE**
Fuzzy neural network	$\langle t_\pi, s_\pi \rangle$	0.53715
Neural network	-	0.5499
Fuzzy rule-based system	$\langle t_{\min}, s_{\max} \rangle$	0.8595
Fuzzy rule-based system	$\langle t_\pi, s_\pi \rangle$	0.8606
Fuzzy rule-based system	$\langle t_{\mathrm{L}}, s_{\mathrm{L}} \rangle$	0.8676

Table 13: RMSE obtained with the different systems presented for the prediction of crop yields.

4 Discussion and suggestions for future work

In this text, we have presented different approaches for the integration of fuzzy logic in AI systems, and have shown how this integration can enhance the interpretability of these systems.

We have described how fuzzy logic can be applied to the implementation of fuzzy rule-based systems, which make predictions based on a predefined set of rules. These systems can be advanced through the definition of additional fuzzy sets for each variable and the adjustment of the rules. This would improve the accuracy, while maintaining the interpretability demonstrated here.

We have presented fuzzy neural networks as a model that combines the advantages of fuzzy logic and classical neural networks. The accuracy achieved by fuzzy neural networks is comparable to that of classical neural networks, due to the optimization of the parameters in the model. Furthermore, the process used to infer the output variable can be explained in terms of rules extracted from the model. This is a significant advantage over classical neural networks in terms of interpretability.

Let us now discuss some observations made during the study, and provide suggestions for future research. First, we have observed that the choice of t-norms and t-conorms can affect the performance of fuzzy neural networks significantly, sometimes even preventing the model from training. Preliminary explorations of this phenomenon have shown that this may be due to the nature of the t-norms and t-conorms in relation to the input data. For instance, the dual pair $\langle t_\pi, s_\pi \rangle$ is not compatible with input membership values that are close to 0. In this case, the composition of t-norms and t-conorms in the network approaches rapidly the value 0, which ultimately becomes the output of the network.

Similar observations have been made for the other dual pairs (see footnote 26). Further investigation of this matter, namely the convergence and divergence of different t-norms and t-conorms, would enable significant improvement of the models presented here. In addition, alternative operators could also be explored. As a starting point, we suggest the use of annihilators, through the partial soft conjunction, as done in [94]. Further discussion can be found in [95].

Second, it has been shown that some of the parameters in the fuzzy neural network are optimized outside the desired range of $[0, 1]$. This may be due to the gradient-based optimization algorithms used for training the model, which are not compatible with constraining the values for the weights. Different algorithms for the optimization of neural networks with constrained weights have been explored in the literature [96–98]. Future research is suggested to explore the use of these algorithms to train fuzzy neural networks. New algorithms specifically designed for this purpose could also be developed.

Third, it has been observed that the last layer of the fuzzy neural network is the only one which has been significantly optimized. This is a commmon phenomenon occurring in the training of classical neural networks. It is often due to the lack of enough data for the training process (in addition to other factors, such as the choice and adjustment of the optimization algorithm). Different solutions to this issue (other than the use of more data, which is sometimes not the most feasible solution) have been explored in the literature [99–102]. Future work may be done to adapt these approaches to fuzzy neural networks.

Last, here we have studied the integration of fuzzy logic operators specifically with feed-forward neural networks (see footnote 15). There are ample possibilities for research on the integration of fuzzy logic with other types of neural networks and deep learning models. Some research has already been done regarding some of these models (see section 1.1 for details), but there is still limited literature regarding the integration of fuzzy logic with the most recent models, such as Transformers and diffusion models. Furthermore, this combination could be explored considering other machine learning models, not necessarily limited to deep learning and neural networks.

In conclusion, we have shown that the integration of fuzzy logic with AI systems can enhance the interpretability of these systems, while still maintaining an accuracy in the prediction comparable to that of classical approaches. The foundations and applications presented here can serve as a starting point for further research in this area, in many different directions.

References

[1] Zadeh, L. A. (1965a). Fuzzy sets. *Information and Control*, *8*(3), 338–353.

[2] Adlassnig, K.-P. (1980). A Fuzzy Logical Model of Computer-Assisted Medical Diagnosis. *Methods of Information in Medicine*, *19*(03), 141–148.

[3] Dagar, P., Jatain, A., & Gaur, D. (2015). Medical diagnosis system using fuzzy logic toolbox. *International Conference on Computing, Communication & Automation (ICCCA)*, 193–197.

[4] John, R., & Innocent, P. (2005). Modeling uncertainty in clinical diagnosis using fuzzy logic. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *35*(6), 1340–1350.

[5] Goetcherian, V. (1980). From binary to grey tone image processing using fuzzy logic concepts. *Pattern Recognition*, *12*(1), 7–15.

[6] Sobrevilla, P., & Montseny, E. (2003). Fuzzy Sets in Computer Vision: An Overview. *Mathware & Soft Computing*, *10*, 71–83.

[7] Amza, C. G., & Cicic, D. T. (2015). Industrial Image Processing Using Fuzzy-logic. *Procedia Engineering*, *100*, 492–498.

[8] Sun, J., Karray, F., Basir, O., et al. (2002). Natural language understanding through fuzzy logic inference and its application to speech recognition. *IEEE International Conference on Fuzzy Systems. FUZZ-IEEE*, *2*, 1120–1125 vol.2.

[9] Carvalho, J. P., Batista, F., & Coheur, L. (2012). A critical survey on the use of Fuzzy Sets in Speech and Natural Language Processing. *IEEE International Conference on Fuzzy Systems*, 1–8.

[10] Alvisi, S., Mascellani, G., Franchini, M., et al. (2006). Water level forecasting through fuzzy logic and artificial neural network approaches. *Hydrology and Earth System Sciences*, *10*(1), 1–17.

[11] Chen, S. X., Gooi, H. B., & Wang, M. Q. (2013). Solar radiation forecast based on fuzzy logic and neural networks. *Renewable Energy*, *60*, 195–201.

[12] Hasan, M., Tsegaye, T., Shi, X., et al. (2008). Model for predicting rainfall by fuzzy set theory using USDA scan data. *Agricultural Water Management*, *95*(12), 1350–1360.

[13] Mahabir, C., Hicks, F. E., & Fayek, A. R. (2003). Application of fuzzy logic to forecast seasonal runoff. *Hydrological Processes*, *17*(18), 3749–3762.

[14] Russell, S. J., & Norvig, P. (2020). *Artificial intelligence: A modern approach, 4th edition*. Pearson.

[15] Crevier, D. (1993). *AI: The Tumultuous History Of The Search For Artificial Intelligence*. Basic Books.

[16] Bělohlávek, R., Dauben, J. W., & Klir, G. J. (2017). *Fuzzy Logic and Mathematics: A Historical Perspective*. Oxford University Press.

[17] McCarthy, J. (1960). Recursive functions of symbolic expressions and their computation by machine, Part I. *Communications of the ACM*, *3*(4), 184–195.

[18] Minsky, M. (1961). Steps toward Artificial Intelligence. *Proceedings of the IRE*, *49*(1), 8–30.

[19] Newell, A., Shaw, J., & Simon, H. (1959). Report on a general problem-solving program. *Proceedings of the International Conference on Information Processing*, 256–264.

[20] Bellman, R. E., & Zadeh, L. A. (1970). Decision-Making in a Fuzzy Environment. *Management Science*, *17*(4), B141–B164.

[21] Buchanan, B. G., & Shortliffe, E. H. (1984). *Rule-based Expert System – The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley.

[22] Waterman, D. A. (1986). *A Guide to Expert Systems*. Addison-Wesley.

[23] Zadeh, L. A. (1973). Outline of a New Approach to the Analysis of Complex Systems and Decision Processes. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-3*(1), 28–44.

[24] Zadeh, L. A. (1983). The role of fuzzy logic in the management of uncertainty in expert systems. *Fuzzy Sets and Systems*, *11*(1), 199–227.

[25] Adlassnig, K.-P., & Kolarz, G. (1982). CADIAG-2: Computer-assisted medical diagnosis using fuzzy subsets. *Approximate Reasoning in Decision Analysis*, 219–247.

[26] Hudson, D. L., & Cohen, M. E. (1987). Fuzzy logic in medical expert systems. *26th IEEE Conference on Decision and Control*, *26*, 337–342.

[27] Kacprzyk, J., & Yager, R. R. (1985). Emergency-Oriented expert systems: A fuzzy approach. *Information Sciences*, *37*(1), 143–155.

[28] Lea, R. N. (1987). Fuzzy Sets And Autonomous Navigation. *Applications of Artificial Intelligence V*, *0786*, 448–452.

[29] Sugeno, M. (1985). *Industrial applications of fuzzy control*. Elsevier Science.

[30] Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. Macmillan.

[31] Gupta, M. M., & Rao, D. H. (1994). On the principles of fuzzy neural networks. *Fuzzy Sets and Systems*, *61*(1), 1–18.

[32] Jang, J.-S. R. (1993). ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, *23*(3), 665–685.

[33] Sun, C.-T., & Jang, J.-S. (1993). A neuro-fuzzy classifier and its applications. *Second IEEE International Conference on Fuzzy Systems*, 94–98 vol.1.

[34] Shann, J. J., & Fu, H. C. (1995). A fuzzy neural network for rule acquiring on fuzzy control systems. *Fuzzy Sets and Systems*, *71*(3), 345–357.

[35] Watanabe, K., Tang, J., Nakamura, M., et al. (1996). A fuzzy-Gaussian neural network and its application to mobile robot control. *IEEE Transactions on Control Systems Technology*, *4*(2), 193–199.

[36] Buckley, J. J., & Hayashi, Y. (1994). Fuzzy neural networks: A survey. *Fuzzy Sets and Systems*, *66*(1), 1–13.

[37] Chun, M.-G., Kwak, K.-C., & Ryu, J.-W. (1999). Application of ANFIS for coagulant dosing process in a water purification plant. *IEEE International Fuzzy Systems*, *3*, 1743–1748.

[38] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[39] Murphy, K. P. (2022). *Probabilistic machine learning: An introduction*. MIT Press.

[40] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

[41] Devlin, J., Chang, M.-W., Lee, K., et al. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv [preprint]*.

[42] Radford, A., Narasimhan, K., Salimans, T., et al. (2018). Improving Language Understanding by Generative Pre-Training. *OpenAI [preprint]*.

[43] Hinton, G., Deng, L., Yu, D., et al. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, *29*(6), 82–97.

[44] Kirillov, A., Mintun, E., Ravi, N., et al. (2023). Segment Anything. *arXiv [preprint]*.

[45] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, 1097–1105.

[46] Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

[47] Levine, S., Pastor, P., Krizhevsky, A., et al. (2016). Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *arXiv [preprint]*.

[48] Ma, J., Sheridan, R. P., Liaw, A., et al. (2015). Deep neural nets as a method for quantitative structure-activity relationships. *Journal of Chemical Information and Modeling*, *55*(2), 263–274.

[49] Angermueller, C., Pärnamaa, T., Parts, L., et al. (2016). Deep learning for computational biology. *Molecular Systems Biology*, *12*(7), 878.

[50] Pimentel, M. A. F., Clifton, D. A., Clifton, L., et al. (2014). A review of novelty detection. *Signal Processing*, *99*, 215–249.

[51] Klir, G. J., & Yuan, B. (1995). *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Pearson.

[52] Guidotti, R., Monreale, A., Ruggieri, S., et al. (2018). A Survey of Methods for Explaining Black Box Models. *ACM Computing Surveys*, *51*(5), 93:1–93:42.

[53] Lin, C.-T., & Lee, C. (1991). Neural-network-based fuzzy logic control and decision system. *IEEE Transactions on Computers*, *40*(12), 1320–1336.

[54] Cpałka, K. (2017). *Design of Interpretable Fuzzy Systems*. Springer International.

[55] Singh, B., Doborjeh, M., Doborjeh, Z., et al. (2023). Constrained neuro fuzzy inference methodology for explainable personalised modelling with applications on gene expression data. *Scientific Reports*, *13*(1), 456.

[56] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., et al. (2017). Building machines that learn and think like people. *The Behavioral and Brain Sciences*, *40*.

[57] Zhang, C., Bengio, S., Hardt, M., et al. (2017). Understanding deep learning requires rethinking generalization. *arXiv [preprint]*.

[58] Zadeh, L. (1996). Fuzzy logic = computing with words. *IEEE Transactions on Fuzzy Systems*, *4*(2), 103–111.

[59] de Campos Souza, P. V. (2020). Fuzzy neural networks and neuro-fuzzy networks: A review the main techniques and applications used in the literature. *Applied Soft Computing*, *92*, 106275.

[60] Popko, E. A., & Weinstein, I. A. (2016). Fuzzy Logic Module of Convolutional Neural Network for Handwritten Digits Recognition. *Journal of Physics: Conference Series*, *738*(1), 012123.

[61] Xi, Z., & Panoutsos, G. (2018). Interpretable Machine Learning: Convolutional Neural Networks with RBF Fuzzy Logic Classification Rules. *International Conference on Intelligent Systems*, 448–454.

[62] Kolman, E., & Margaliot, M. (2009). Extracting symbolic knowledge from recurrent neural networks—A fuzzy logic approach. *Fuzzy Sets and Systems*, *160*(2), 145–161.

[63] Subathra, B., & Radhakrishnan, T. K. (2012). Recurrent Neuro Fuzzy and Fuzzy Neural Hybrid Networks: A Review. *Instrumentation Science & Technology*, *40*(1), 29–50.

[64] Al-Hmouz, R., Pedrycz, W., Balamash, A., et al. (2019). Logic-driven autoencoders. *Knowledge-Based Systems*, *183*, 104874.

[65] Chauhan, N., & Choi, B.-J. (2019). Denoising Approaches Using Fuzzy Logic and Convolutional Autoencoders for Human Brain MRI Image. *International Journal of Fuzzy Logic and Intelligent Systems*, *19*(3), 135–139.

[66] Nguyen, R., Singh, S. K., & Rai, R. (2023). FuzzyGAN: Fuzzy generative adversarial networks for regression tasks. *Neurocomputing*, *525*, 88–110.

[67] Berenji, H. R. (1992). A reinforcement learning—based architecture for fuzzy logic control. *International Journal of Approximate Reasoning*, *6*(2), 267–292.

[68] He, C., Liu, S., & Han, S. (2020). A Fuzzy Logic Reinforcement Learning-Based Routing Algorithm For Flying Ad Hoc Networks. *International Conference on Computing, Networking and Communications (ICNC)*, 987–991.

[69] Yung, N. H. C., & Ye, C. (1999). An intelligent mobile vehicle navigator based on fuzzy logic and reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *29*(2), 314–321.

[70] Omisore, M. O., Samuel, O. W., & Atajeromavwo, E. J. (2017). A Genetic-Neuro-Fuzzy inferential model for diagnosis of tuberculosis. *Applied Computing and Informatics*, *13*(1), 27–37.

[71] Reddy, G. T., Reddy, M. P. K., Lakshmanna, K., et al. (2020). Hybrid genetic algorithm and a fuzzy logic classifier for heart disease diagnosis. *Evolutionary Intelligence*, *13*(2), 185–196.

[72] Chimatapu, R., Hagras, H., Starkey, A., et al. (2018). Explainable AI and Fuzzy Logic Systems. *Theory and Practice of Natural Computing*, 3–20.

[73] Freitas, A. A. (2014). Comprehensible classification models: A position paper. *ACM SIGKDD Explorations Newsletter*, *15*(1), 1–10.

[74] Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., et al. (2019). Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI. *arXiv [preprint]*.

[75] Lipton, Z. C. (2017). The Mythos of Model Interpretability. *arXiv [preprint]*.

[76] Zadeh, L. A. (1965b). Fuzzy Sets and Systems. *Proceedings of the Symposium on System Theory*, *15*.

[77] Goodstein, R. L. (1963). *Boolean algebra*. Pergamon, Macmillan.

[78] Stoll, R. R. (1979). *Set Theory and Logic*. Dover Publications.

[79] Phuong, N. H., & Kreinovich, V. (2001). Fuzzy logic and its applications in medicine. *International Journal of Medical Informatics*, *62*(2), 165–173.

[80] Smithson, M. J., & Verkuilen, J. (2006). *Fuzzy Set Theory: Applications in the Social Sciences*. SAGE Publications.

[81] Zimmermann, H.-J. (2001). *Fuzzy Set Theory And Its Applications, 4Th Edition*. Springer.

[82] Zimmermann, H.-J. (2010). Fuzzy set theory. *Wiley Interdisciplinary Reviews: Computational Statistics*, *2*(3), 317–332.

[83] Hájek, P. (1998). *Metamathematics of Fuzzy Logic*. Springer.

[84] Jayaram, M. A., & Marad, N. (2012). Fuzzy Inference Systems for Crop Yield Prediction. *Journal of Intelligent Systems*, *21*(4), 363–372.

[85] Li, R., Li, M., Ashraf, U., et al. (2019). Exploring the Relationships Between Yield and Yield-Related Traits for Rice Varieties Released in China From 1978 to 2017. *Frontiers in Plant Science*, *10*.

[86] Murphy, K. P. (2023). *Probabilistic machine learning: Advanced topics*. MIT Press.

[87] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

[88] LeNail, A. (2019). NN-SVG: Publication-Ready Neural Network Architecture Schematics. *Journal of Open Source Software*.

[89] Kingma, D. P., & Ba, J. (2017). Adam: A method for stochastic optimization. *arXiv [preprint]*.

[90] Paszke, A., Gross, S., Massa, F., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *arXiv [preprint]*.

[91] Hirota, K., & Pedrycz, W. (1994). OR/AND neuron in modeling fuzzy set connectives. *IEEE Transactions on Fuzzy Systems*, *2*(2), 151–161.

[92] Costa, V., Dellunde, P., & Falomir, Z. (2019). The logical style painting classifier based on Horn clauses and explanations (ℓ-SHE). *Logic Journal of the IGPL*, *29*(1), 96–119.

[93] Ruder, S. (2017). An overview of gradient descent optimization algorithms. *arXiv [preprint]*.

[94] Costa, V. (2020). The art painting style Classifier based on Logic Aggregators and qualitative colour Descriptors (C-LAD). *Proceedings of the 9th European Starting AI Researchers' Symposium 2020*, *2655*.

[95] Dujmovic, J. (2018). *Soft Computing Evaluation Logic: The LSP Decision Method and Its Applications*. Wiley.

[96] Ding, S., Su, C., & Yu, J. (2011). An optimizing BP neural network algorithm based on genetic algorithm. *Artificial Intelligence Review*, *36*(2), 153–162.

[97] Sexton, R. S., Dorsey, R. E., & Johnson, J. D. (1999). Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing. *European Journal of Operational Research*, *114*(3), 589–601.

[98] Whitley, D., Starkweather, T., & Bogart, C. (1990). Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, *14*(3), 347–361.

[99] Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, *6*(60).

[100] Liu, P., Wang, X., Xiang, C., et al. (2020). A Survey of Text Data Augmentation. *2020 International Conference on Computer Communication and Network Security (CCNS)*, 191–195.

[101] Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). A survey of transfer learning. *Journal of Big Data*, *3*(9).

[102] Ohno, H. (2020). Auto-encoder-based generative models for data augmentation on regression problems. *Soft Computing*, *24*(11), 7999–8009.

A Code for the implementations of the systems

This appendix includes extracts of the code written in Python for the implementations of the models presented in this text. The code shown here includes the relevant parts for the implementation of the systems. The full code, with outputs, figures and evaluations (including the code to produce the figures and the evaluations), is available in the form of Jupyter Notebooks and Python script files in the Github repository found online at: https://github.com/loredanasandu/tfg-fuzzy-logic-artificial-intelligence.

A.1 Data processing

```
import pandas as pd

# Read the raw data from the csv file
df = pd.read_csv('data/rice-yield-raw-data.csv', header=1)

# Select the features of interest and rename them
df_clean = df[['Ecotype', 'Panicle number (million/ha)', 'Growth period (d)',
               'Yield (t/ha)']].copy()
df_clean.rename(columns={'Yield (t/ha)': 'yield', 'Growth period (d)': 'growth',
                         'Panicle number (million/ha)': 'panicle'}, inplace=True)

# Remove rows with missing values
df_clean.dropna(axis=0, inplace=True)

# Select the rows corresponding to the Japonica inbred type of rice plant, and remove
# the 'Ecotype' column (no longer needed)
df_clean = df_clean[df_clean['Ecotype'] == 'Japonica inbred']
df_clean.drop(columns=df_clean.columns[0], inplace=True)

# Remove the rows with outliers
# Remove values lower than or equal to the 0.01 percentile
df_clean = df_clean[df_clean['yield'] > df_clean['yield'].quantile(0.01)]
df_clean = df_clean[df_clean['growth'] > df_clean['growth'].quantile(0.01)]
df_clean = df_clean[df_clean['panicle'] > df_clean['panicle'].quantile(0.01)]

# Remove values higher than or equal to the 0.99 percentile
df_clean = df_clean[df_clean['yield'] < df_clean['yield'].quantile(0.99)]
df_clean = df_clean[df_clean['growth'] < df_clean['growth'].quantile(0.99)]
df_clean = df_clean[df_clean['panicle'] < df_clean['panicle'].quantile(0.99)]

# Save the processed data to a new csv file
df_clean.to_csv('data/rice-yield-clean-data.csv', index=False)
```

A.2 Fuzzy rule-based system

Definition of generic membership functions

```
  def triangular_membership_function(x:float, a:float, b:float, c:float) -> float:
      """
      Triangular membership function.

      Args:
          x (float): input value
          a (float): leftmost point of the triangle
          b (float): middle point of the triangle
          c (float): rightmost point of the triangle
```

```
    Returns:
        float: membership value between 0 and 1
    """
    try:
        if x < a:
            return 0
        elif a <= x <= b:
            return (x - a) / (b - a)
        elif b <= x <= c:
            return (c - x) / (c - b)
        else:
            return 0
    except ZeroDivisionError:
        return 0

def left_trapezoidal_membership_function(x:float, a:float, b:float) -> float:
    """
    Left trapezoidal membership function.

    Args:
        x (float): input value
        a (float): leftmost point of the trapezoid
        b (float): rightmost point of the trapezoid

    Returns:
        float: membership value between 0 and 1
    """
    try:
        if x <= a:
            return 1
        elif a <= x <= b:
            return (b - x) / (b - a)
        else:
            return 0
    except ZeroDivisionError:
        return 0

def right_trapezoidal_membership_function(x:float, a:float, b:float) -> float:
    """
    Right trapezoidal membership function.

    Args:
        x (float): input value
        a (float): leftmost point of the trapezoid
        b (float): rightmost point of the trapezoid

    Returns:
        float: membership value between 0 and 1
    """
    try:
        if x <= a:
            return 0
        elif a <= x <= b:
            return (x - a) / (b - a)
        else:
            return 1
    except ZeroDivisionError:
        return 0
```

Definition of t-norms and t-conorms

```python
import numpy as np

# --- T-norms ---

def t_norm_min(x: float, y: float) -> float:
    """
    Minimum t-norm.
    Defined as the minimum of x and y.

    Args:
        x (float): first value, membership value between 0 and 1
        y (float): second value, membership value between 0 and 1

    Returns:
        float: minimum of x and y, membership value between 0 and 1
    """
    return np.min([x, y], axis=0)


def t_norm_product(x: float, y: float) -> float:
    """
    Product t-norm.
    Defined as the product of x and y.

    Args:
        x (float): first value, membership value between 0 and 1
        y (float): second value, membership value between 0 and 1

    Returns:
        float: product of x and y, membership value between 0 and 1
    """
    return x * y


def t_norm_lukasiewicz(x: float, y: float) -> float:
    """
    Lukasiewicz t-norm.
    Defined as the maximum of 0 and x + y - 1.

    Args:
        x (float): first value, membership value between 0 and 1
        y (float): second value, membership value between 0 and 1

    Returns:
        float: maximum of 1 and x + y, membership value between 0 and 1
    """
    return np.max([np.zeros(x.shape), x + y - 1], axis=0)


# --- T-conorms ---

def t_conorm_max(x: float, y: float) -> float:
    """
    Maximum t-conorm.
    Defined as the maximum of x and y.

    Args:
        x (float): first value, membership value between 0 and 1
        y (float): second value, membership value between 0 and 1

    Returns:
        float: maximum of x and y, membership value between 0 and 1
    """
    return np.max([x, y], axis=0)
```

```
  def t_conorm_sum(x: float, y: float) -> float:
      """
      Sum t-conorm.
      Defined as x + y - x * y.

      Args:
          x (float): first value, membership value between 0 and 1
          y (float): second value, membership value between 0 and 1

      Returns:
          float: sum of x and y, membership value between 0 and 1
      """
      return x + y - x * y

  def t_conorm_lukasiewicz( x: float, y: float) -> float:
      """
      Lukasiewicz t-conorm.
      Defined as the minimum of 1 and x + y.

      Args:
          x (float): first value, membership value between 0 and 1
          y (float): second value, membership value between 0 and 1

      Returns:
          float: minimum of 1 and x + y, membership value between 0 and 1
      """
      return np.min([np.ones(x.shape), x + y], axis=0)
```

Definition of the fuzzy rule-based system

```
import numpy as np
import pandas as pd


class YieldPredictionSystem():
  """Fuzzy rule-based system for rice yield prediction."""
  def __init__(self,
               data: pd.DataFrame,     # columns: panicle, growth, yield
               t_norm: str = 'min',
               t_conorm: str = 'max') -> None:
      """
      Initialize the yield prediction system.

      Args:
          data (pd.DataFrame): data to be used for training and testing the system
          t_norm (str, optional): t-norm to be used for evaluating rules.
              Defaults to 'min'.
          t_conorm (str, optional): t-conorm to be used for evaluating rules.
              Defaults to 'max'.
      """
      # Check t-norm and t-conorm
      assert t_norm in ['min', 'product', 'lukasiewicz']
      assert t_conorm in ['max', 'sum', 'lukasiewicz']

      # Check data
      assert 'panicle' in data.columns and data['panicle'].dtype == 'float64'
      assert 'growth' in data.columns and data['growth'].dtype == 'float64'
      assert 'yield' in data.columns and data['yield'].dtype == 'float64'
      assert data.columns.size == 3

      # Save percentiles for panicle, growth and yield
      self.panicle_datapoints = {
          'min': data['panicle'].min(),
```

```
            'pct10': data['panicle'].quantile(0.1),
            'pct25': data['panicle'].quantile(0.25),
            'pct50': data['panicle'].quantile(0.5),
            'pct75': data['panicle'].quantile(0.75),
            'pct90': data['panicle'].quantile(0.9),
            'max': data['panicle'].max()
        }
        self.growth_datapoints = {
            'min': data['growth'].min(),
            'pct10': data['growth'].quantile(0.1),
            'pct25': data['growth'].quantile(0.25),
            'pct50': data['growth'].quantile(0.5),
            'pct75': data['growth'].quantile(0.75),
            'pct90': data['growth'].quantile(0.9),
            'max': data['growth'].max()
        }
        self.yield_datapoints = {
            'min': data['yield'].min(),
            'pct10': data['yield'].quantile(0.1),
            'pct25': data['yield'].quantile(0.25),
            'pct50': data['yield'].quantile(0.5),
            'pct75': data['yield'].quantile(0.75),
            'pct90': data['yield'].quantile(0.9),
            'max': data['yield'].max(),
        }

        # Set t-norm and t-conorm functions
        if t_norm == 'min':
            self.t_norm = t_norm_min
        elif t_norm == 'product':
            self.t_norm = t_norm_product
        elif t_norm == 'lukasiewicz':
            self.t_norm = t_norm_lukasiewicz

        if t_conorm == 'max':
            self.t_conorm = t_conorm_max
        elif t_conorm == 'sum':
            self.t_conorm = t_conorm_sum
        elif t_conorm == 'lukasiewicz':
            self.t_conorm = t_conorm_lukasiewicz

        # Initialize the data and membership functions
        self.data = data
        self.set_membership_functions()

        self.rules = None


    # --- Membership functions ---

    def set_membership_functions(self) -> None:
        """
        Set membership functions for panicle, growth and yield. We use the 10th, 25th,
        50th, 75th and 90th percentiles to define the piecewise membership functions.
        """
        # Membership functions for panicle
        self.mu_VLP = lambda x: left_trapezoidal_membership_function(
            x, a=self.panicle_datapoints['pct10'], b=self.panicle_datapoints['pct25'])
        self.mu_LP = lambda x: triangular_membership_function(
            x, a=self.panicle_datapoints['pct10'], b=self.panicle_datapoints['pct25'],
            c=self.panicle_datapoints['pct50'])
        self.mu_MP = lambda x: triangular_membership_function(
            x, a=self.panicle_datapoints['pct25'], b=self.panicle_datapoints['pct50'],
            c=self.panicle_datapoints['pct75'])
        self.mu_HP = lambda x: triangular_membership_function(
          x, a=self.panicle_datapoints['pct50'], b=self.panicle_datapoints['pct75'],
          c=self.panicle_datapoints['pct90'])
        self.mu_VHP = lambda x: right_trapezoidal_membership_function(
```

```
            x, a=self.panicle_datapoints['pct75'], b=self.panicle_datapoints['pct90'])

        # Membership functions for growth
        self.mu_VLG = lambda x: left_trapezoidal_membership_function(
            x, a=self.growth_datapoints['pct10'], b=self.growth_datapoints['pct25'])
        self.mu_LG = lambda x: triangular_membership_function(
            x, a=self.growth_datapoints['pct10'], b=self.growth_datapoints['pct25'],
            c=self.growth_datapoints['pct50'])
        self.mu_MG = lambda x: triangular_membership_function(
            x, a=self.growth_datapoints['pct25'], b=self.growth_datapoints['pct50'],
            c=self.growth_datapoints['pct75'])
        self.mu_HG = lambda x: triangular_membership_function(
            x, a=self.growth_datapoints['pct50'], b=self.growth_datapoints['pct75'],
            c=self.growth_datapoints['pct90'])
        self.mu_VHG = lambda x: right_trapezoidal_membership_function(
            x, a=self.growth_datapoints['pct75'], b=self.growth_datapoints['pct90'])

        # Membership functions for yield
        self.mu_VLY = lambda x: left_trapezoidal_membership_function(
            x, a=self.yield_datapoints['pct10'], b=self.yield_datapoints['pct25'])
        self.mu_LY = lambda x: triangular_membership_function(
            x, a=self.yield_datapoints['pct10'], b=self.yield_datapoints['pct25'],
            c=self.yield_datapoints['pct50'])
        self.mu_MY = lambda x: triangular_membership_function(
            x, a=self.yield_datapoints['pct25'], b=self.yield_datapoints['pct50'],
            c=self.yield_datapoints['pct75'])
        self.mu_HY = lambda x: triangular_membership_function(
            x, a=self.yield_datapoints['pct50'], b=self.yield_datapoints['pct75'],
            c=self.yield_datapoints['pct90'])
        self.mu_VHY = lambda x: right_trapezoidal_membership_function(
            x, a=self.yield_datapoints['pct75'], b=self.yield_datapoints['pct90'])


    # --- Fuzzify input ---

    def fuzzify_input(self) -> None:
        """
        Fuzzify panicle and growth.
        """
        # Fuzzify panicle
        self.data['VLP'] = self.data['panicle'].apply(self.mu_VLP)
        self.data['LP'] = self.data['panicle'].apply(self.mu_LP)
        self.data['MP'] = self.data['panicle'].apply(self.mu_MP)
        self.data['HP'] = self.data['panicle'].apply(self.mu_HP)
        self.data['VHP'] = self.data['panicle'].apply(self.mu_VHP)

        # Fuzzify growth
        self.data['VLG'] = self.data['growth'].apply(self.mu_VLG)
        self.data['LG'] = self.data['growth'].apply(self.mu_LG)
        self.data['MG'] = self.data['growth'].apply(self.mu_MG)
        self.data['HG'] = self.data['growth'].apply(self.mu_HG)
        self.data['VHG'] = self.data['growth'].apply(self.mu_VHG)


    # --- Evaluate rules ---

    def evaluate_rules(self, rules: list, data: pd.DataFrame) -> None:
        """
        Evaluate two-to-one rules. Rules are given by tuples of the form
            (operator, panicle_fuzzy_set, growth_fuzzy_set, yield_fuzzy_set)
        and they are expressions of the form:
            IF panicle is <panicle_fuzzy_set> <operator> growth is <growth_fuzzy_set>
                THEN yield is <yield_fuzzy_set>

        Example: ('AND', "LP", "LG", "LY") means:
            IF panicle is LP AND growth is LG THEN yield is LY

        Args:
```

```
            rules (list): list of rules in the form of tuples (operator, panicle_fuzzy_set,
              growth_fuzzy_set, yield_fuzzy_set)
            data (pd.DataFrame): data to be used for evaluating the rules

        Raises:
            ValueError: if the operator is not 'AND' or 'OR'
        """
        # Initialize the aggregated membership values for yield categories
        data['pred_VLY'] = 0.0
        data['pred_LY'] = 0.0
        data['pred_MY'] = 0.0
        data['pred_HY'] = 0.0
        data['pred_VHY'] = 0.0

        # Evaluate each rule
        for rule in rules:
            # Get rule parameters
            operator = rule[0]
            panicle_fuzzy_set, growth_fuzzy_set, yield_fuzzy_set = rule[1], rule[2], rule[3]

            # Evaluate rule
            if operator == 'AND':
                yield_membership = self.t_norm(
                    data[panicle_fuzzy_set], data[growth_fuzzy_set])
                data['pred_' + yield_fuzzy_set] = self.t_conorm(
                    data['pred_' + yield_fuzzy_set], yield_membership)

            elif operator == 'OR':
                yield_membership = self.t_conorm(
                    data[panicle_fuzzy_set], data[growth_fuzzy_set])
                data['pred_' + yield_fuzzy_set] = self.t_conorm(
                    data['pred_' + yield_fuzzy_set], yield_membership)

            else:
                raise ValueError('Invalid operator')


    # --- Defuzzify yield ---

    def defuzzify_yield(self) -> None:
        """
        Defuzzify yield using the centroid method.
        """
        x = np.linspace(self.yield_datapoints['min'], self.yield_datapoints['max'], 1000)
        y_VLY = self.data['pred_VLY'].apply(
          lambda pred: [self.mu_VLY(x_i) if self.mu_VLY(x_i) < pred else pred for x_i in x])
        y_LY = self.data['pred_LY'].apply(
          lambda pred: [self.mu_LY(x_i) if self.mu_LY(x_i) < pred else pred for x_i in x])
        y_MY = self.data['pred_MY'].apply(
          lambda pred: [self.mu_MY(x_i) if self.mu_MY(x_i) < pred else pred for x_i in x])
        y_HY = self.data['pred_HY'].apply(
          lambda pred: [self.mu_HY(x_i) if self.mu_HY(x_i) < pred else pred for x_i in x])
        y_VHY = self.data['pred_VHY'].apply(
          lambda pred: [self.mu_VHY(x_i) if self.mu_VHY(x_i) < pred else pred for x_i in x])

        # For each x_i save the largest y_i
        y = np.array([np.array([np.max([y_VLY[j][i], y_LY[j][i], y_MY[j][i], y_HY[j][i],
                      y_VHY[j][i]]) for i in range(len(x))]) for j in range(len(self.data))])

        # Calculate the centroid
        self.data['predicted_crisp_yield'] = np.sum(x * y, axis=1) / np.sum(y, axis=1)
```

Main code for testing the system

See the Github repository specified above for a Jupyter notebook with output tables and figures. We include here the most relevant main code for testing the system, which builds on the code shown above.

```
import pandas as pd
import numpy as np

# Load the data
df_system = pd.read_csv('data/rice-yield-clean-data.csv')

# Initialize the fuzzy system.
# The t-norm and t-conorm can be chosen among ('min', 'max'), ('product', 'sum')
# and ('lukasiewicz', 'lukasiewicz').
fuzzy_system = YieldPredictionSystem(data=df_system, t_norm='product', t_conorm='sum')

# Fuzzify the input data
fuzzy_system.fuzzify_input()

# Evaluate rules
fuzzy_system.rules = [
            ['AND', 'VLP', 'VLG', 'VLY'],
            ['AND', 'VLP', 'LG', 'LY'],
            ['AND', 'VLP', 'MG', 'LY'],
            ['AND', 'VLP', 'HG', 'MY'],
            ['AND', 'VLP', 'VHG', 'MY'],

            ['AND', 'LP', 'VLG', 'VLY'],
            ['AND', 'LP', 'LG', 'LY'],
            ['AND', 'LP', 'MG', 'LY'],
            ['AND', 'LP', 'HG', 'MY'],
            ['AND', 'LP', 'VHG', 'MY'],

            ['AND', 'MP', 'VLG', 'LY'],
            ['AND', 'MP', 'LG', 'MY'],
            ['AND', 'MP', 'MG', 'MY'],
            ['AND', 'MP', 'HG', 'HY'],
            ['AND', 'MP', 'VHG', 'HY'],

            ['AND', 'HP', 'VLG', 'LY'],
            ['AND', 'HP', 'LG', 'MY'],
            ['AND', 'HP', 'MG', 'MY'],
            ['AND', 'HP', 'HG', 'HY'],
            ['AND', 'HP', 'VHG', 'HY'],

            ['AND', 'VHP', 'VLG', 'MY'],
            ['AND', 'VHP', 'LG', 'MY'],
            ['AND', 'VHP', 'MG','HY'],
            ['AND', 'VHP', 'HG', 'VHY'],
            ['AND', 'VHP', 'VHG', 'VHY'],
        ]

fuzzy_system.evaluate_rules(fuzzy_system.rules, fuzzy_system.data)

# Defuzzify yield
fuzzy_system.defuzzify_yield()

# Calculate RMSE of the predicted crisp yield
RMSE_predicted_vs_real = np.sqrt(np.mean((fuzzy_system.data['yield'] - \
                                          fuzzy_system.data['predicted_crisp_yield'])**2))
```

A.3 Neural network

```
import pandas as pd
import numpy as np

import torch
import torch.nn as nn
import torch.optim as optim

import copy


# Load the data
df_nn = pd.read_csv('data/rice-yield-clean-data.csv')

# Fix the random seed (this is to ensure the results can be reproduced)
np.random.seed(2)
torch.manual_seed(2)

# Define the model
model = nn.Sequential(
    # Layer 1
    nn.Linear(2, 8),
    nn.ReLU(),

    # Layer 2
    nn.Linear(8, 4),
    nn.ReLU(),

    # Output layer
    nn.Linear(4, 1),
    nn.ReLU(),
)

# Define the loss function (Mean Squared Error) and the optimizer (Adam)
loss_function = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Define input and output variables
X = df_nn[['panicle', 'growth']].values
y = df_nn['yield'].values
y = y.reshape(y.shape[0],1)

# Scale the input data
X = (X - X.mean(axis=0)) / X.std(axis=0)

# Train-test split (80% training, 20% testing)
X_train, X_test = np.split(X, [int(.8 *len(X))])
y_train, y_test = np.split(y, [int(.8 *len(y))])

# Convert the data from numpy arrays to PyTorch tensors
X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.FloatTensor(y_train)
y_test = torch.FloatTensor(y_test)

# Save the best model for later analysis
best_mse = np.inf   # initialize to infinity
best_weights = None

# Save the losses for later visualization
test_losses = []
train_losses = []

# Train and test the model
n_epochs = 100      # number of epochs
for epoch in range(n_epochs):
```

```
    model.train()
    epoch_train_losses = []

    # Training loop
    for i in range(len(X_train)):
        # take a sample
        X_sample = X_train[i]
        y_sample = y_train[i]

        # Forward pass
        y_pred = model(X_sample)
        loss = loss_function(y_pred, y_sample)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()

        # Update weights
        optimizer.step()

        # Save loss for later visualization
        epoch_train_losses.append(float(loss))

    # Testing
    model.eval()
    y_pred = model(X_test)
    test_loss = float(loss_function(y_pred, y_test))

    # Save losses and best model for later analysis
    mean_train_loss = np.mean(epoch_train_losses)
    train_losses.append(mean_train_loss)
    test_losses.append(test_loss)
    if test_loss < best_mse:
        best_mse = test_loss
        best_weights = copy.deepcopy(model.state_dict())
    last_mse = test_loss
    last_weights = copy.deepcopy(model.state_dict())

    print('Epoch %d/%d' % \ (epoch+1, n_epochs))
    print(f"Train loss: {mean_train_loss:.6f}")
    print(f"Test loss: {test_loss:.6f}")
```

Now we can plot the losses achieved in each iteration of the training system (variables `train_losses` and `test_losses`). We can also use either the best model (variable `best_weights`) or the model optimized in the last epoch (variable `last_weights`) to predict the yield. The weights of the model can be loaded with `model.load_state_dict(weights)`, where `weights` is the variable containing the desired weights. See the Github repository specified above for the full code.

A.4 Fuzzy neural network

Loading and fuzzifying the data

```
import pandas as pd
import numpy as np

import torch
import torch.nn as nn
import torch.optim as optim

import copy
```

```
# The t-norm and t-conorm can be chosen among ('min', 'max'), ('product', 'sum'),
# ('lukasiewicz', 'lukasiewicz')
T_NORM = 'product'
T_CONORM = 'sum'

# Load the data
df_fnn = pd.read_csv('data/rice-yield-clean-data.csv')

# We will use the YieldPredictionSystem class to fuzzify the input and defuzzify the output
fuzzy_system_fnn = YieldPredictionSystem(data=df_fnn, t_norm=T_NORM, t_conorm=T_CONORM)

# Fuzzify the input
fuzzy_system_fnn.fuzzify_input()

# Fuzzify the output data
fuzzy_system_fnn.data['real_VLY'] = fuzzy_system_fnn.data['yield'].apply(fuzzy_system_fnn.mu_VLY)
fuzzy_system_fnn.data['real_LY'] = fuzzy_system_fnn.data['yield'].apply(fuzzy_system_fnn.mu_LY)
fuzzy_system_fnn.data['real_MY'] = fuzzy_system_fnn.data['yield'].apply(fuzzy_system_fnn.mu_MY)
fuzzy_system_fnn.data['real_HY'] = fuzzy_system_fnn.data['yield'].apply(fuzzy_system_fnn.mu_HY)
fuzzy_system_fnn.data['real_VHY'] = fuzzy_system_fnn.data['yield'].apply(fuzzy_system_fnn.mu_VHY)

# Include negations of the fuzzy sets
not_mu = lambda x: 1 - x
fuzzy_system_fnn.data['not_VLP'] = fuzzy_system_fnn.data['VLP'].apply(not_mu)
fuzzy_system_fnn.data['not_LP'] = fuzzy_system_fnn.data['LP'].apply(not_mu)
fuzzy_system_fnn.data['not_MP'] = fuzzy_system_fnn.data['MP'].apply(not_mu)
fuzzy_system_fnn.data['not_HP'] = fuzzy_system_fnn.data['HP'].apply(not_mu)
fuzzy_system_fnn.data['not_VHP'] = fuzzy_system_fnn.data['VHP'].apply(not_mu)
fuzzy_system_fnn.data['not_VLG'] = fuzzy_system_fnn.data['VLG'].apply(not_mu)
fuzzy_system_fnn.data['not_LG'] = fuzzy_system_fnn.data['LG'].apply(not_mu)
fuzzy_system_fnn.data['not_MG'] = fuzzy_system_fnn.data['MG'].apply(not_mu)
fuzzy_system_fnn.data['not_HG'] = fuzzy_system_fnn.data['HG'].apply(not_mu)
fuzzy_system_fnn.data['not_VHG'] = fuzzy_system_fnn.data['VHG'].apply(not_mu)

# Define the input and output variables for the fuzzy neural network
X = fuzzy_system_fnn.data[['VLP', 'LP', 'MP', 'HP', 'VHP',
                           'VLG', 'LG', 'MG', 'HG', 'VHG',
                           'not_VLP', 'not_LP', 'not_MP', 'not_HP', 'not_VHP',
                           'not_VLG', 'not_LG', 'not_MG', 'not_HG', 'not_VHG']].values
y = fuzzy_system_fnn.data[['real_VLY', 'real_LY', 'real_MY', 'real_HY', 'real_VHY']].values
```

Definition of the fuzzy neural network

We define special functions to compute the t-norm and t-conorm for the and-neurons and or-neurons neurons, with an input vector and a matrix of weights. These operations will be used later in the definition of the fuzzy neurons. These operators have been tested before incorporating them in the fuzzy neural network.

```
  if T_NORM == 'min':
      # t-norm for the AND layer, returns a vector
      t_norm_and_layer = lambda x: torch.min(x, dim=0).values

      # t-norm for the OR layer, returns a matrix
      t_norm_or_layer = lambda x, y: torch.min(x, y)

  if T_CONORM == 'max':
      # t-conorm for the AND layer, returns a matrix
      t_conorm_and_layer = lambda x, y: torch.max(x, y)

      # t-conorm for the OR layer, returns a vector
      t_conorm_or_layer = lambda x: torch.max(x, dim=0).values
```

```
if T_NORM == 'product':
    # t-norm for the AND layer, returns a vector
    t_norm_and_layer = lambda x: torch.prod(x, dim=0)

    # t-norm for the OR layer, returns a matrix
    t_norm_or_layer = lambda x, y: torch.mul(x, y)

if T_CONORM == 'sum':
    # t-conorm for the AND layer, returns a matrix
    t_conorm_and_layer = lambda x, y: x + y - torch.mul(x, y)

    # sum t-conorm
    t_conorm_sum = lambda x, y: x + y - torch.mul(x, y)

    # t-conorm for the OR layer, returns a vector
    def t_conorm_or_layer(x):
        while True:
            if x.shape[0] == 1:
                x.reshape(-1)
                break
            elif x.shape[0] == 2:
                x = t_conorm_sum(x[0], x[1])
                x.reshape(-1)
                break
            else:
                x = torch.cat((torch.stack(
                      [t_conorm_sum(x[0][i], x[1][i]) for i in range(x.shape[1])]
                    ).reshape(1,-1), x[2:]))
        return x


if T_NORM == 'lukasiewicz':
    # lukaiewicz t-norm
    t_norm_lukasiewicz = lambda x, y: torch.max(torch.zeros(x.shape), x + y - 1)

    # t-norm for the AND layer, returns a vector
    def t_norm_and_layer(x):
        while True:
            if x.shape[0] == 1:
                x.reshape(-1)
                break
            elif x.shape[0] == 2:
                x = t_norm_lukasiewicz(x[0], x[1])
                x.reshape(-1)
                break
            else:
                x = torch.cat((torch.stack(
                      [t_norm_lukasiewicz(x[0][i], x[1][i]) for i in range(x.shape[1])]
                    ).reshape(1,-1), x[2:]))
        return x

    # t-norm for the OR layer, returns a matrix
    t_norm_or_layer = lambda x, y: torch.max(torch.zeros(x.shape), x + y - 1)

if T_CONORM == 'lukasiewicz':
    # t-conorm for the AND layer, returns a matrix
    t_conorm_and_layer = lambda x, y: torch.min(torch.ones(x.shape), x + y)

    # lukasiewicz t-conorm
    t_conorm_lukasiewicz = lambda x, y: torch.min(torch.ones(x.shape), x + y)

    # t-conorm for the OR layer, returns a vector
    def t_conorm_or_layer(x):
        while True:
            if x.shape[0] == 1:
                x.reshape(-1)
                break
```

```
            elif x.shape[0] == 2:
                x = t_conorm_lukasiewicz(x[0], x[1])
                x.reshape(-1)
                break
            else:
                x = torch.cat((torch.stack(
                      [t_conorm_lukasiewicz(x[0][i], x[1][i]) for i in range(x.shape[1])]
                    ).reshape(1,-1), x[2:]))
        return x
```

We now define the and-layers (layers with and-neurons) and or-layers (layers with or-neurons) that will be used in the fuzzy neural network.

```
# Fix the random seed (this is to ensure the results can be reproduced)
np.random.seed(2)
torch.manual_seed(2)

class AndLayer(nn.Module):
    """AND layer of the fuzzy neural network."""
    def __init__(self, in_units, out_units):
        """
        Initialize the layer.

        Args:
            in_units (int): Number of inputs
            out_units (int): Number of outputs
        """
        super().__init__()
        self.weight = nn.Parameter(torch.rand((in_units, out_units)))

    def forward(self, X):
        """
        Forward pass of the layer.

        Args:
            X (torch.Tensor): Input tensor of shape (in_units)

        Returns:
            torch.Tensor: Output tensor of shape (out_units)
        """

        with torch.no_grad():
            # reshape X to a matrix of shape (in_units, 1) (column vector)
            X = X.reshape(-1, 1)
            # repeat X along columns, so that every column is equal to previous X
            X = X.expand_as(self.weight)

        # t_conorm of weigths and X (this gives, for each column of weights, the
        # t_conorm between the input X and the column of weights)
        OR_matrix = t_conorm_and_layer(X, self.weight)

        # t_norm of each column of OR_matrix
        AND_vector = t_norm_and_layer(OR_matrix)

        return AND_vector

class OrLayer(nn.Module):
    """OR layer of the fuzzy neural network."""
    def __init__(self, in_units, out_units):
        """
        Initialize the layer.

        Args:
            in_units (int): Number of inputs
            out_units (int): Number of outputs
```

```
        """
        super().__init__()
        self.weight = nn.Parameter(torch.rand((in_units, out_units)))

    def forward(self, X):
        """
        Forward pass of the layer.

        Args:
            X (torch.Tensor): Input tensor of shape (in_units)

        Returns:
            torch.Tensor: Output tensor of shape (out_units)
        """

        with torch.no_grad():
            # reshape X to a matrix of shape (in_units, 1) (column vector)
            X = X.reshape(-1, 1)
            # repeat X along columns, so that every column is equal to previous X
            X = X.expand_as(self.weight)

        # t_norm of weigths and X (this gives, for each column of weights, the
        # t_norm between the input X and the column of weights)
        AND_matrix = t_norm_or_layer(X, self.weight)

        # t_conorm of each column of AND_matrix
        OR_vector = t_conorm_or_layer(AND_matrix)

        return OR_vector
```

Last, we define the fuzzy neural network, using the and-layers and or-layers defined above. The network is specifically for the use case of the rice yield prediction shown in the text. The hyperparameters (number of layers, number of neurons per layer, input and output shape, etc.) can be changed to fit other use cases.

```
class FuzzyNeuralNetwork(nn.Module):
    """Fuzzy Neural Network."""
    def __init__(self):
        """
        Initialize the network.
        """
        super(FuzzyNeuralNetwork, self).__init__()
        self.l1 = AndLayer(20,20)
        self.l2 = OrLayer(20,10)
        self.l3 = OrLayer(10,5)

    def forward(self,x):
        """
        Forward pass of the network.

        Args:
            x (torch.Tensor): Input tensor of shape (20)

        Returns:
            torch.Tensor: Output tensor of shape (5)
        """
        out= self.l1(x)
        out= self.l2(out)
        out= self.l3(out)
        return(out)
```

Training and testing the fuzzy neural network

```
# Define the model
model = FuzzyNeuralNetwork()

# Define the loss function (Mean Squared Error) and the optimizer (Adam)
loss_function = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.1)

# Train-test split (80% training, 20% testing)
X_train, X_test= np.split(X, [int(.8 *len(X))])
y_train, y_test= np.split(y, [int(.8 *len(y))])

# Convert the data from numpy arrays to PyTorch tensors
X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.FloatTensor(y_train)
y_test = torch.FloatTensor(y_test)

# Save the best model for later analysis
best_mse = np.inf   # initialize to infinity
best_weights = None

# Save losses for later visualization
test_losses = []
train_losses = []

# Train and test the model
n_epochs = 25      # number of epochs
for epoch in range(n_epochs):
    model.train()
    epoch_train_losses = []

    # Training loop
    for i in range(len(X_train)):
        # take a sample
        X_sample = X_train[i]
        y_sample = y_train[i]

        # Forward pass
        y_pred = model(X_sample)
        loss = loss_function(y_pred, y_sample)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()

        # Update weights
        optimizer.step()

        # Save loss for later visualization
        epoch_train_losses.append(float(loss))

    # Testing
    model.eval()
    y_pred = torch.stack([model(X_test[i]) for i in range(X_test.shape[0])])
    test_loss = float(loss_function(y_pred, y_test))

    # Save losses and best model for later analysis
    mean_train_loss = np.mean(epoch_train_losses)
    train_losses.append(mean_train_loss)
    test_losses.append(test_loss)
    if test_loss < best_mse:
        best_mse = test_loss
        best_weights = copy.deepcopy(model.state_dict())
    last_mse = test_loss
```

```
    last_weights = copy.deepcopy(model.state_dict())

    print('Epoch %d/%d' % \ (epoch+1, n_epochs))
    print(f"Train loss: {mean_train_loss:.6f}")
    print(f"Test loss: {test_loss:.6f}")
```

Similarly to the neural network, we can plot the losses and predict the yield with either the best model (variable `best_weights`) or the model optimized in the last epoch (variable `last_weights`) to predict the yield. The weights of the model can be loaded with `model.load_state_dict(weights)`, with `weights` the variable containing the desired weights. The output predicted by the model is defuzzified with `fuzzy_system_fnn.defuzzify_yield()`. See the Github repository specified above for the full code.