Fuzzy Without Fuzzy: Why Fuzzy-Related Aggregation Techniques Are Often Better Even in Situations Without True Fuzziness

Hung T. Nguyen[1], Vladik Kreinovich[2], François Modave[2], and Martine Ceberio[2]

[1] Department of Mathematical Sciences, New Mexico State University, Las Cruces, NM 88003, USA hunguyen@nmsu.edu
[2] Department of Computer Science, University of Texas at El Paso, El Paso, TX 79968, USA, vladik@utep.edu, fmodave@utep.edu, mceberio@cs.utep.edu

Summary. Fuzzy techniques have been originally invented as a methodology that transforms the knowledge of experts formulated in terms of natural language into a precise computer-implementable form. There are many successful applications of this methodology to situations in which expert knowledge exist, the most well known is an application to fuzzy control.

In some cases, fuzzy methodology is applied even when no expert knowledge exists: instead of trying to approximate the unknown control function by splines, polynomials, or by any other traditional approximation technique, researchers try to approximate it by guessing and tuning the expert rules. Surprisingly, this approximation often works fine, especially in such application areas as control and multi-criteria decision making.

In this paper, we give a mathematical explanation for this phenomenon.

1 Introduction

Fuzzy techniques: a brief reminder. Fuzzy techniques have been originally invented as a methodology that transforms the knowledge of experts formulated in terms of natural language into a precise computer-implementable form. There are many successful applications of this methodology to situations in which expert knowledge exist, the most well known is an application to fuzzy control; see, e.g., [6, 8, 18].

Universal approximation results. A guarantee of success comes from the fact that fuzzy systems are *universal approximators* in the sense that for every continuous function $f(x_1, \ldots, x_n)$ and for every $\varepsilon > 0$, there exists a set of rules for which the corresponding input-output function is ε-close to f; see, e.g., [1, 8, 9, 11, 12, 16, 18, 19, 22, 23, 25] and references therein.

Fuzzy methodology is sometimes successful without any fuzzy expert knowledge. In some cases, fuzzy methodology is applied even when no expert knowledge exists: instead of trying to approximate the unknown control function by splines, polynomials, or by any other traditional approximation technique, researchers try to approximate it by guessing and tuning the expert rules. Surprisingly, this approximation often works fine.

Similarly, fuzzy-type aggregation functions like OWA or Choquet integrals often work better than quadratic functions in multi-criteria decision making.

What we plan to do. In this paper, we give a mathematical explanation for these phenomena, and we show that approximation by using fuzzy methodology is indeed (in some reasonable sense) the best.

Comment. In this paper, we build upon our preliminary results published in [13, 15, 17].

2 Use of Fuzzy Techniques in Non-Fuzzy Control: A Justification

In many practical applications, data processing speed is important. We have mentioned that one of the main applications of fuzzy methodology is to intelligent control.

In applications to automatic control, the computer must constantly compute the current values of control. The value of the control depends on the state of the controlled object (called *plant* in control theory). So, to get a high quality control, we must measure as many characteristics $x_1, \ldots, x_n$ of the current state as we can. The more characteristics we measure, the more numbers we have to process, so, the more computation steps we must perform. The results of these computations must be ready in no time, before we start the next round of measurements. So, automatic control, especially high-quality automatic control, is a real-time computation problem with a serious time pressure.

Parallel computing is an answer. A natural way to increase the speed of the computations is to perform computations *in parallel* on several processors. To make the computations really fast, we must divide the algorithm into parallelizable steps, each of which requires a small amount of time.

What are these steps?

The fewer variables, the faster. As we have already mentioned, the main reason why control algorithms are computationally complicated is that we must process many inputs. For example, controlling a car is easier than controlling a plane, because the plane (as a 3-D object) has more characteristics to take care of, more characteristics to measure and hence, more characteristics to process. Controlling a space shuttle, especially during the lift-off and landing,

is even a more complicated task, usually performed by several groups of people who control the trajectory, temperature, rotation, etc. In short, the more numbers we need to process, the more complicated the algorithm. Therefore, if we want to decompose our algorithm into fastest possible modules, we must make each module to process as few numbers as possible.

Functions of one variable are not sufficient. Ideally, we should only use the modules that compute functions of one variable. However, if we only have functions of one variables (i.e., procedures with one input and one output), then, no matter how we combine them, we will always end up with functions of one variable. Since our ultimate goal is to compute the control function $u = f(x_1, \ldots, x_n)$ that depends on many variables $x_1, \ldots, x_n$, we must therefore enable our processors to compute at least one function of two or more variables.

What functions of two variables should we choose?

Choosing functions of two or more variables. Inside the computer, each function is represented as a sequence of hardware implemented operations. The fastest functions are those that are computed by a single hardware operation. The basic hardware supported operations are: arithmetic operations $a + b$, $a - b$, $a \cdot b$, a/b, and $\min(a, b)$ and $\max(a, b)$. The time required for each operation, crudely speaking, corresponds to the number of bits operations that have to be performed:

- Division is done by successive multiplication, comparison and subtraction (basically, in the same way as we do it manually), so, it is a much slower operation than $-$.
- Multiplication is implemented as a sequence of additions (again, basically in the same manner as we do it manually), so it is much slower than $+$.
- $-$ and $+$ are usually implemented in the same way. To add two n-bit binary numbers, we need n bit additions, and also potentially, n bit additions for carries. Totally, we need about $2n$ bit operations.
- min of two n-bit binary numbers can be done in n binary operations: we compare the bits from the highest to the lowest, and as soon as they differ, the number that has 0 as opposed to 1 is the desired minimum: e.g., the minimum of 0.10101 and 0.10011 is 0.10011, because in the third bit, this number has 0 as opposed to 1.
- Similarly, max is an n-bit operation.

So, the fastest possible functions of two variables are min and max. Similarly fast is computing the minimum and maximum of several (more than two) real numbers. Therefore, we will choose these functions for our control-oriented computer.

Summarizing the above-given analysis, we can conclude that our computer will contain modules of two type:

- modules that compute functions of one variable;
- modules that compute min and max of two or several numbers.

How to combine these modules? We want to combine these modules in such a way that the resulting computations are as fast as possible. The time that is required for an algorithm is crudely proportional to the number of sequential steps that it takes. We can describe this number of steps in clear geometric terms:

- at the beginning, the input numbers are processed by some processors; these processors form the *first layer* of computations;
- the results of this processing may then go into different processors, that form the *second layer*;
- the results of the second layer of processing go into the *third layer*,
- etc.

In these terms, the fewer layers the computer has, the faster it is.

So, we would like to combine the processors into the smallest possible number of layers.

Now, we are ready for the formal definitions.

Definition and the main result. Let us first give an inductive definition of what it means for a function to be computable by a k-layer computer.

Definition 1.

- *We say that a function $f(x_1, \ldots, x_n)$ is* computable by a 1-layer computer *if either $n = 1$, or the function f coincides with* min *or with* max.
- *Let $k \geq 1$ be an integer. We say that a function $f(x_1, \ldots, x_n)$ is* computable by a $(k+1)$-layer computer *if one of the following three statements is true:*
  - *$f = g(h(x_1, \ldots, x_n))$, where g is a function of one variable, and $h(x_1, \ldots, x_n)$ is computable by a k-layer computer;*
  - *$f = \min(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$, where all functions g_i are computed by a k-layer computer;*
  - *$f = \max(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$, where all functions g_i are computed by a k-layer computer.*

Comment. A computer is a finite-precision machine, so, the results of the computations are never absolutely precise. Also, a computer is limited in the size of its numbers. So, we can only compute a function approximately, and only on a limited range. Therefore, when we say that we can compute an arbitrary function, we simply mean that for an arbitrary range T, for an arbitrary continuous function $f : [-T, T]^n \to R$, and for an arbitrary accuracy $\varepsilon > 0$, we can compute a function that is ε-close to f on the given range. In this sense, we will show that not every function can be computed on a 2-layer computer, but that 3 layers are already sufficient.

Proposition 1. *There exist real numbers T and $\varepsilon > 0$, and a continuous function $f : [-T, T]^n \to R$ such that no function ε-close to f on $[-T, T]^n$ can be computed on a 2-layer computer.*

Comment. To make the text more readable, we present both proofs in the last section. However, we will make one comment here. The function that will be proved to be not computable on a 2-layer computer is not exotic at all: it is $f(x_1, x_2) = x_1 + x_2$ on the domain $[-1, 1]^2$, and Proposition 1 holds for $\varepsilon = 0.4$.

Theorem 1. *For every real numbers T and $\varepsilon > 0$, and for every continuous function $f : [-T, T]^n \to R$, there exists a function $\widetilde{f}$ that is ε-close to f on $[-T, T]^n$ and that is computable on a 3-layer computer.*

Comment. In other words, *functions computed by a 3-layer computer are universal approximators.*

Relation to fuzzy control. As we will see from the proof, the approximating function $\widetilde{f}$ is of the type $\max(A_1, \ldots, A_m)$, where

$$A_j = \min(f_{j1}(x_1), \ldots, f_{jn}(x_n).$$

These functions correspond the so-called *fuzzy control* [6, 8, 18]: Indeed, let us define

$$U = \max_{i,j,x_i \in [-T,T]} |f_{ji}(x_i)|,$$

and

$$\mu_{ji}(x_i) = \frac{f_{ji}(x_i) - (-U)}{U - (-U)}.$$

Let us now assume that the rules base that describes the expert recommendations for control consists of exactly two rules:

- "if one of the conditions C_j is true, then $u = U$";
- "else, $u = -U$",

where each condition C_j means that the following n conditions are satisfied:

- x_1 satisfies the property C_{j1} (described by a membership function $\mu_{j1}(x_1)$);
- x_2 satisfies the property C_{j2} (described by a membership function $\mu_{j2}(x_2)$);
- ...
- x_n satisfies the property C_{jn} (described by a membership function $\mu_{jn}(x_n)$).

In logical terms, the condition C for $u = U$ has the form

$$(C_{11}\& \ldots \& C_{1n}) \vee \ldots \vee (C_{k1}\& \ldots \& C_{kn}).$$

If we use min for &, and max for $\vee$ (these are the simplest choices in fuzzy control methodology), then the degree μ_C with which we believe in a condition $C = C_1 \vee \ldots \vee C_k$ can be expressed as:

$$\mu_C = \max[\min(\mu_{11}(x_1), \ldots, \mu_{1n}), \ldots, \min(\mu_{k1}, \ldots, \mu_{kn})].$$

Correspondingly, the degree of belief in a condition for $u = -U$ is $1 - \mu_C$. According to fuzzy control methodology, we must use a *defuzzification* to determine the actual control, which in this case leads to the choice of

$$u = \frac{U \cdot \mu_C + (-U) \cdot (1 - \mu_C)}{\mu_C + (1 - \mu_C)}.$$

Because of our choice of μ_{ji}, one can easily see that this expression coincides exactly with the function $\max(A_1, \ldots, A_m)$, where

$$A_j = \min(f_{j1}(x_1), \ldots, f_{jn}(x_n)).$$

So, we get exactly the expressions that stem from the fuzzy control methodology.

Conclusion. Since our 3-layer expression describes the fastest possible computation tool, we can conclude that *for control problems, the fastest possible universal computation scheme corresponds to using fuzzy methodology.*

This result explains why fuzzy methodology is sometimes used (and used successfully) without any expert knowledge being present, as an extrapolation tool for the (unknown) function.

Comment. We have considered *digital* parallel computers. If we use *analog* processors instead, then min and max stop being the simplest functions. Instead, the sum is the simplest: if we just join the two wires together, then the resulting current is equal to the sum of the two input currents. In this case, if we use a sum (and more general, linear combination) instead of min and max, 3-layer computers are also universal approximators; the corresponding computers correspond to *neural networks* [10].

Proof of Proposition 1.

0°. Let us proof (by reduction to a contradiction) that if a function $\widetilde{f}(x_1, x_2)$ is $0.4-$close to $f(x_1, x_2) = x_1 + x_2$ on $[-1, 1]^2$, then $\widetilde{f}$ cannot be computed on a 2-layer computer. Indeed, suppose that it is. Then, according to the Definition, the function $\widetilde{f}(x_1, x_2)$ is of one of the following three forms:

- $g(h(x_1, x_2))$, where h is computable on a 1-layer computer;
- $\min(g_1(x_1, x_2), \ldots, g_m(x_1, x_2))$, where all the functions g_i are computable on a 1-layer computer;
- $\max(g_1(x_1, x_2), \ldots, g_m(x_1, x_2))$, where all the functions g_i are computable on a 1-layer computer.

Let us show case-by-case that all these three cases are impossible.

1°. In the first case, $\widetilde{f}(x_1, x_2) = g(h(x_1, x_2))$, where h is computable on a 1-layer computer. Be definition, this means that h is either a function of one variable, or min, or max. Let us consider all these three sub-cases.

1.1°. If $\widetilde{f}(x_1, x_2) = g(h(x_1))$, then the function $\widetilde{f}$ depends only on x_1. In particular,

$$\widetilde{f}(0, -1) = \widetilde{f}(0, 1). \tag{1}$$

But since $\widetilde{f}$ is ε-close to $f(x_1 + x_2) = x_1 + x_2$, we get

$$\widetilde{f}(0,-1) \le f(0,-1) + \varepsilon = -1 + 0.4 = -0.6,$$

and

$$\widetilde{f}(0,1) \ge f(0,1) - \varepsilon = 1 - 0.4 > 0.6 > -0.6.$$

So, $\widetilde{f}(0,-1) \le -0.6 < \widetilde{f}(0,1)$, hence, $\widetilde{f}(0,-1) \ne \widetilde{f}(0,1)$, which contradicts to (1). So, this sub-case is impossible. Similarly, it is impossible to have h depending only on x_2.

1.2°. Let us consider the sub-case when $\widetilde{f}(x_1,x_2) = g(\min(x_1,x_2))$. In this sub-case,

$$\widetilde{f}(-1,-1) = g(\min(-1,-1)) = g(-1) = g(\min(-1,1)) = \widetilde{f}(-1,1),$$

and

$$\widetilde{f}(-1,-1) = \widetilde{f}(-1,1). \tag{2}$$

But

$$\widetilde{f}(-1,-1) \le f(-1,-1) + \varepsilon = -2 + 0.4 = -1.6,$$

and

$$\widetilde{f}(-1,1) \ge f(-1,1) - \varepsilon = 0 - 0.4 = -0.4 > -1.6,$$

so, the equality (2) is also impossible.

1.3°. Let us now consider the sub-case $\widetilde{f}(x_1,x_2) = g(\max(x_1,x_2))$. In this sub-case,

$$\widetilde{f}(-1,1) = g(\max(-1,1)) = g(1) = g(\max(1,1)) = \widetilde{f}(1,1),$$

and

$$\widetilde{f}(-1,1) = \widetilde{f}(1,1). \tag{3}$$

But

$$\widetilde{f}(-1,1) \le f(-1,1) + \varepsilon = 0 + 0.4 = 0.4,$$

and

$$\widetilde{f}(1,1) \ge f(1,1) - \varepsilon = 2 - 0.4 = 1.6 > 0.4,$$

so, the equality (3) is also impossible.

2°. In the second case, $\widetilde{f}(x_1,x_2) = \min(g_1(x_1,x_2),\ldots,g_m(x_1,x_2))$, where all the functions g_i are computable on a 1-layer computer. For this case, the impossibility follows from the following sequence of steps:

2.1°. If one of the functions g_i is of the type $\min(x_1,x_2)$, then we can rewrite

$$\min(g_1,\ldots,g_{i-1},\min(x_1,x_2),g_{i+1},\ldots,g_m)$$

as

$$\min(g_1,\ldots,g_{i-1},g_i^{(1)},g_i^{(2)},g_{i+1},\ldots,g_m),$$

where $g^{(i)}(x_1, x_2) = x_i$ is a function that is clearly computable on a 1-layer computer. After we make such transformations, we get an expression for $\widetilde{f}$ that only contains max and functions of one variable.

2.2°. Let us show that this expression cannot contain max. Indeed, if it does, then

$$\widetilde{f}(x_1, x_2) = \min(\ldots, \max(x_1, x_2)) \le \max(x_1, x_2).$$

In particular, $\widetilde{f}(1, 1) \le \max(1, 1) = 1$. But we must have

$$\widetilde{f}(1, 1) \ge f(1, 1) - \varepsilon = 2 - 0.4 = 1.6 > 1.$$

The contradiction shows that max cannot be one of the functions g_i.

2.3°. So, each function g_i depends only on one variable. If all of them depend on one and the same variable, say, x_1, then the entire function $\widetilde{f}$ depends only on one variable, and we have already proved (in the proof of the first case) that it is impossible. So, some functions g_i depend on x_1, and some of the functions g_i depend on x_2. Let us denote by $h_1(x_1)$ the minimum of all functions g_i that depend on x_1, and by $h_2(x_2)$, the minimum of all the functions g_i that depend on x_2. Then, we can represent $\widetilde{f}$ as $\widetilde{f}(x_1, x_2) = \min(h_1(x_1), h_2(x_2))$.

2.4°. To get a contradiction, let us first take $x_1 = 1$ and $x_2 = 1$. Then,

$$\widetilde{f}(1, 1) = \min(h_1(1), h_2(1)) \ge f(1, 1) - \varepsilon = 2 - 0.4 = 1.6.$$

Since the minimum of the two numbers is ≥ 1.6, we can conclude that each of them is ≥ 1.6, i.e., that $h_1(1) \ge 1.6$ and $h_2(1) \ge 1.6$. For $x_1 = 1$ and $x_2 = -1$, we have

$$\widetilde{f}(1, -1) = \min(h_1(1), h_2(-1)) \le f(1, -1) + \varepsilon = 0.4.$$

Since $h_1(1) \ge 1.6$, we conclude that $\widetilde{f}(1, -1) = h_2(-1)$. From

$$\widetilde{f}(1, -1) \ge f(1, -1) - \varepsilon = -0.4, \tag{4}$$

we can now conclude that $h_2(-1) \ge -0.4$. Similarly, one can prove that $h_1(-1) \ge -0.4$. Hence,

$$\widetilde{f}(-1, -1) = \min(h_1(-1), h_2(-1)) \ge -0.4.$$

But

$$\widetilde{f}(-1, -1) \le f(-1, -1) + \varepsilon = -2 + 0.4 = -1.6 < -0.4 :$$

a contradiction with (4).

The contradiction shows that the second case is also impossible.

3°. In the third case, $\widetilde{f}(x_1, x_2) = \max(g_1(x_1, x_2), \ldots, g_m(x_1, x_2))$, where all the functions g_i are computable on a 1-layer computer. For this case, the impossibility (similarly to the second case) follows from the following sequence of steps:

3.1°. If one of the functions g_i is of the type $\max(x_1, x_2)$, then we can rewrite

$$\max(g_1, \ldots, g_{i-1}, \max(x_1, x_2), g_{i+1}, \ldots, g_m)$$

as

$$\max(g_1, \ldots, g_{i-1}, g_i^{(1)}, g_i^{(2)}, g_{i+1}, \ldots, g_m),$$

where $g^{(i)}(x_1, x_2) = x_i$ is a function that is clearly computable on a 1-layer computer. After we make such transformations, we get an expression for $\widetilde{f}$ that only contains min and functions of one variable.

3.2°. Let us show that this expression cannot contain min. Indeed, if it does, then

$$\widetilde{f}(x_1, x_2) = \max(\ldots, \min(x_1, x_2)) \geq \min(x_1, x_2).$$

In particular,

$$\widetilde{f}(-1, -1) \geq \min(-1, -1) = -1.$$

But we must have

$$\widetilde{f}(-1, -1) \leq f(-1, -1) + \varepsilon = -2 + 0.4 = -1.6 < -1.$$

The contradiction shows that min cannot be one of the functions g_i.

3.3°. So, each function g_i depends only on one variable. If all of them depend on one and the same variable, say, x_1, then the entire function $\widetilde{f}$ depends only on one variable, and we have already proved (in the proof of the first case) that it is impossible. So, some functions g_i depend on x_1, and some of the functions g_i depend on x_2. Let us denote by $h_1(x_1)$ the maximum of all functions g_i that depend on x_1, and by $h_2(x_2)$, the maximum of all the functions g_i that depend on x_2. Then, we can represent $\widetilde{f}$ as

$$\widetilde{f}(x_1, x_2) = \max(h_1(x_1), h_2(x_2)).$$

3.4°. To get a contradiction, let us first take $x_1 = -1$ and $x_2 = -1$. Then,

$$\widetilde{f}(-1, -1) = \max(h_1(-1), h_2(-1)) \leq f(-1, -1) + \varepsilon = -2 + 0.4 = -1.6.$$

Since the maximum of the two numbers is ≤ -1.6, we can conclude that each of them is ≤ -1.6, i.e., that $h_1(-1) \leq -1.6$ and $h_2(-1) \leq -1.6$. For $x_1 = 1$ and $x_2 = -1$, we have

$$\widetilde{f}(1, -1) = \max(h_1(1), h_2(-1)) \geq f(1, -1) - \varepsilon = -0.4.$$

Since $h_2(-1) \leq -1.6$, we conclude that $\widetilde{f}(1, -1) = h_1(1)$. From

$$\widetilde{f}(1, -1) \leq f(1, -1) + \varepsilon = 0.4,$$

we can now conclude that $h_1(1) \le 0.4$. Similarly, one can prove that $h_2(1) \le 0.4$. Hence,

$$\widetilde{f}(1,1) = \max(h_1(1), h_2(1)) \ge 0.4. \tag{5}$$

But

$$\widetilde{f}(1,1) \ge f(1,1) - \varepsilon = 2 - 0.4 = 1.6 > 0.4,$$

which contradicts to (5).

The contradiction shows that the third case is also impossible.

4°. In all there cases, we have shown that the assumption that $\widetilde{f}$ can be computed on a 2-layer computer leads to a contradiction. So, $\widetilde{f}$ cannot be thus computed. Q.E.D.

Proof of Theorem 1. Since the function f is continuous, there exists a $\delta > 0$ such that if $|x_i - y_i| \le \delta$, then

$$|f(x_1, \ldots, x_n) - f(y_1, \ldots, y_n)| \le \varepsilon.$$

Let us mark the grid points on the grid of size δ, i.e., all the points for which each coordinate $x_1, \ldots, x_n$ has the form $q_i \cdot \delta$ for integer q_i (i.e., we mark the points with coordinates 0, $\pm\delta$, $\pm 2\delta$, ..., $\pm T$).

On each coordinate, we thus mark $\approx 2T/\delta$ points. So, totally, we mark $\approx (2T/\delta)^n$ grid points. Let us denote the total number of grid points by k, and the points themselves by $P_j = (x_{j1}, \ldots, x_{jn})$, $1 \le j \le k$.

By m_f, let us denote the minimum of f:

$$m_f = \min_{x_1 \in [-T,T], \ldots, x_n \in [-T,T]} f(x_1, \ldots, x_n).$$

For each grid point P_j, we will form piece-wise linear functions $f_{ji}(x_i)$ as follows:

- if $|x_i - x_{ji}| \le 0.6 \cdot \delta$, then $f_{ji}(x_i) = f(P_j) (\ge m_f)$;
- if $|x_i - x_{ji}| \ge 0.7 \cdot \delta$, then $f_{ji}(x_i) = m_f$;
- if $0.6 \cdot \delta \le |x_i - x_{ji}| \le 0.7 \cdot \delta$, then

$$f_{ji}(x_i) = m_f + (f(P_j) - m_f) \cdot \frac{0.7 \cdot \delta - |x_i - x_{ji}|}{0.7 \cdot \delta - 0.6 \cdot \delta}.$$

Let us show that for these functions f_{ji}, the function

$$\widetilde{f}(x_1, \ldots, x_n) = \max(A_1, \ldots, A_m),$$

where

$$A_j = \min(f_{j1}(x_1), \ldots, f_{jn}(x_n)),$$

is ε-close to f.

To prove that, we will prove the following two inequalities:

- For all $x_1, \ldots, x_n$, we have $\widetilde{f}(x_1, \ldots, x_n) \ge f(x_1, \ldots, x_n) - \varepsilon$.

- For all $x_1, \ldots, x_n$, we have $\widetilde{f}(x_1, \ldots, x_n) \le f(x_1, \ldots, x_n) + \varepsilon$.

Let us first prove the first inequality. Assume that we have a point $(x_1, \ldots, x_n)$. For every $i = 1, \ldots, n$, by q_i, we will denote the integer that is the closest to x_i/δ. Then,

$$|x_i - q_i \cdot \delta| \le 0.5 \cdot \delta.$$

These values q_i determine a grid point $P_j = (x_{j1}, \ldots, x_{jn})$ with coordinates $x_{ji} = q_i \cdot \delta$. For this j, and for every i,

$$|x_i - x_{ji}| \le 0.5 \cdot \delta < 0.6 \cdot \delta,$$

therefore, by definition of f_{ji}, we have $f_{ji}(x_i) = f(P_j)$. Hence,

$$A_j = \min(f_{j1}(x_1), \ldots, f_{jn}(x_n)) = \min(f(P_j), \ldots, f(P_j)) = f(P_j).$$

Therefore,

$$\widetilde{f}(x_1, .., x_n) = \max(A_1, \ldots, A_m) \ge A_j = f(P_j).$$

But since $|x_{ji} - x_i| \le 0.5 \cdot \delta < \delta$, by the choice of δ, we have

$$|f(x_1, \ldots, x_n) - f(P_j)| \le \varepsilon.$$

Therefore, $f(P_j) \ge f(x_1, \ldots, x_n) - \varepsilon$, and hence,

$$\widetilde{f}(x_1, \ldots, x_n) \ge f(P_j) \ge f(x_1, \ldots, x_n) - \varepsilon.$$

Let us now prove the second inequality. According to our definition of f_{ji}, the value of $f_{ji}(x_i)$ is always in between m_f and P_j, and this value is different from m_f only for the grid points P_j for which $|x_{ji} - x_i| \le 0.7 \cdot \delta$. The value

$$A_j = \min(f_{j1}(x_1), \ldots, f_{jn}(x_n))$$

is thus different from m only if all the values $f_{ji}(x_i)$ are different from m, i.e., when $|x_{ji} - x_i| \le 0.7 \cdot \delta$ for all i. For this grid point, $|x_{ji} - x_i| \le 0.7 \cdot \delta < \delta$; therefore,

$$|f(P_j) - f(x_1, \ldots, x_n)| \le \varepsilon$$

and hence, $f(P_j) \le f(x_1, \ldots, x_n) + \varepsilon$. By definition of f_{ji}, we have $f_{ji}(x_i) \le f(P_j)$. Since this is true for all i, we have

$$A_j = \min(f_{j1}(x_1), \ldots, f_{jn}(x_n)) \le f(P_j) \le f(x_1, \ldots, x_n) + \varepsilon.$$

For all other grid points P_j, we have

$$A_j(x_1, \ldots, x_n) = m_f$$

for a given $(x_1, \ldots, x_n)$. Since m_f has been defined as the minimum of f, we have

$$A_j = m_f \le f(x_1, \ldots, x_n) < f(x_1, \ldots, x_n) + \varepsilon.$$

So, for all grid points, we have

$$A_j \le f(x_1, \ldots, x_n) + \varepsilon,$$

and therefore,

$$\widetilde{f}(x_1, \ldots, x_n) = \max(A_1, \ldots, A_m) \le f(x_1, \ldots, x_n) + \varepsilon.$$

The second inequality is also proven.

So, both inequalities are true, and hence, $\widetilde{f}$ is ε-close to f. The theorem is proven.

3 Fuzzy-Type Aggregation in Multi-Criteria Decision Making: A Problem

A similar situation occurs in multi-criterion decision making. To describe the problem, let us briefly explain what multi-criteria decision making is about.

One of the main purposes of Artificial Intelligence in general is to incorporate a large part of human intelligent reasoning and decision-making into a computer-based systems, so that the resulting intelligent computer-based systems help users in making rational decisions. In particular, to help a user make a decision among a large number of alternatives, an intelligent decision-making systems should select a small number of these alternatives – alternatives which are of the most potential interest to the user.

For example, with so many possible houses on the market, it is not realistically possible to have a potential buyer inspect all the house sold in a given city. Instead, a good realtor tries to find out the buyer's preferences and only show him or her houses that more or less fit these preferences. It would be great to have an automated system for making similar pre-selections.

To be able to make this selection, we must elicit the information about the user preferences.

In principle, we can get a full picture of the user preferences by asking the user to compare and/or rank all possible alternatives. Such a complete description of user preferences may be sometimes useful, but in decision making applications, such an extensive question-asking defeats the whole purpose of intelligent decision-making systems – to avoid requiring that the the user make a large number of comparisons.

The existing approach to this problem is called *multi-criteria decision making* (MCDM). The main idea behind this approach is that each alternative is characterized by the values of different parameters. For example, the buyer's selection of a house depends on the house's size, on its age, on its geographical location, on the number of bedrooms and bathrooms, etc. The idea is to elicit preferences corresponding to each of these parameters, and then to combine

these single-parameter preferences into a reasonable model for describing the user's choice.

In the standard decision making theory, preferences are characterized by assigning, to each alternative, a numerical value called its *utility*. In these terms, the multi-criteria decision making approach means that we try to combine single-variable utility values $u_1(x_1), \ldots, u_n(x_n)$ characterizing the user's preferences over individual parameters $x_1, \ldots, x_n$ into a utility value $u(x_1, \ldots, x_n)$ that characterizes the utility of an alternative described by the values $(x_1, \ldots, x_n)$.

In the first approximation, it makes sense simply to add the individual utility values with appropriate weights, i.e., to consider linear aggregation

$$u(x_1, \ldots, x_n) = w_1 \cdot u_1(x_1) + \ldots + w_n \cdot u_n(x_n).$$

In many practical situations, linear aggregation works well, but in some cases, it leads to counterintuitive conclusions. For example, when selecting a house, a user can assign certain weights to all the parameters characterizing different houses, but the user may also has absolute limitations: e.g., a user with kids may want a house with at least two bedrooms, and no advantages in location and price would entice her to buy a one-bedroom house. To describe such reasonable preferences, we must therefore go beyond linear aggregation functions.

From the purely mathematical viewpoint, the inadequacy of a linear model is a particular example of a very typical situation. Often, when we describe the actual dependence between the quantities in physics, chemistry, engineering, etc., a linear expressions $y = c_0 + c_1 \cdot x_1 + \ldots + c_n \cdot x_n$ is a very good first approximation (at least locally), but to get a more accurate approximations, we must take non-linearity into account. In mathematical applications to physics, engineering, etc., there is a standard way to take non-linearity into account: if a linear approximation is not accurate enough, a natural idea is to use a quadratic approximation $y \approx a_0 + \sum\limits_{i=1}^{n} c_i \cdot x_i + \sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} c_{ij} \cdot x_i \cdot x_j$; if the quadratic approximation is not sufficient accurate, we can use a cubic approximation, etc.; see, e.g., [4].

At first glance, it seems reasonable to apply a similar idea to multi-criteria decision making and consider quadratic aggregation functions

$$u \stackrel{\text{def}}{=} u(x_1, \ldots, x_n) = u_0 + \sum_{i=1}^{n} w_i \cdot u_i(x_i) + \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot u_i(x_i) \cdot u_j(x_j).$$

Surprisingly, in contrast to physics and engineering applications, quadratic approximation do not work as well as approximations based on the use of piece-wise linear functions, such as the OWA operation $u = w_1 \cdot u_{(1)} + \ldots + w_n \cdot u_{(n)}$, where $u_{(1)} = \max(u_1(x_1), \ldots, u_n(x_n))$ is the largest of n utility values $u_i(x_i)$, $u_{(2)}$ is the second largest, $\ldots$, and $u_{(n)} = \min(u_1(x_1), \ldots, u_n(x_n))$ is the smallest of n utility values; see, e.g., [24].

In our own research, we have applied OWA and we have also applied similar piece-wise linear operations (based on the so-called Choquet integral [5]), and we also got good results – better than quadratic approximations; see, e.g., [2] and references therein. Similar results have been obtained by others. For quite some time, why piece-wise approximations are better than quadratic ones remains a mystery to us – and to many other researchers whom we asked this question. Now, we finally have an answer to this question – and this answer is presented in the current paper.

Thus, the paper provides a new justification of the use of piece-wise aggregation operations in multi-criteria decision making – a justification that explains why these aggregation operations are better than the (seemingly more natural) quadratic ones.

4 Standard Decision Making Theory: A Brief Reminder

To explain our answer to the long-standing puzzle, we need to recall the properties of the utility functions. The needed properties of utility functions are described in this section. Readers who are already well familiar with the standard decision making theory (and with the corresponding properties of utility functions) can skip this section and proceed directly to the next one.

To be able to describe decisions, we must have a numerical scale for describing preferences. The traditional decision making theory (see, e.g., [7, 14, 20]) starts with an observation that such a scale can be naturally obtained by using probabilities. Specifically, to design this scale, we select two alternatives:

- a very negative alternative A_0; e.g., an alternative in which the decision maker loses all his money (and/or loses his health as well), and
- a very positive alternative A_1; e.g., an alternative in which the decision maker wins several million dollars.

Based on these two alternatives, we can, for every value $p \in [0, 1]$, consider a randomized alternative $L(p)$ in which we get A_1 with probability p and A_0 with probability $1 - p$.

(It should be mentioned that in the standard decision making theory, randomized alternatives like $L(p)$ are also (somewhat misleadingly) called *lotteries*. This name comes from the fact that a lottery is one of the few real-life examples of randomized outcomes with known probabilities.)

In the two extreme cases $p = 0$ and $p = 1$, the randomized alternative $L(p)$ turns into one of the original alternatives: when $p = 1$, we get the favorable alternative A_1 (with probability 1), and when $p = 0$, we get the unfavorable alternative A_0. In general, the larger the probability p of the favorable alternative A_1, the more preferable is the corresponding randomized alternative $L(p)$. Thus, the corresponding randomized alternatives ("lotteries") $L(p)$ form a continuous 1-D scale ranging from the very negative alternative A_0 to the very positive alternative A_1.

So, it is reasonable to gauge the preference of an arbitrary alternative A by comparing it to different alternatives $L(p)$ from this scale until we find A's place on this scale, i.e., the value $p \in [0,1]$ for which, to this decision maker, the alternative A is equivalent to $L(p)$: $L(p) \sim A$. This value is called the *utility* $u(A)$ of the alternative A in the standard decision making theory.

In our definition, the numerical value of the utility depends on the selection of the alternatives A_0 and A_1: e.g., A_0 is the alternative whose utility is 0 and A_1 is the alternative whose utility is 1. What if we use a different set of alternatives, e.g., $A_0' < A_0$ and $A_1' > A_1$?

Let A be an arbitrary alternative between A_0 and A_1, and let $u(A)$ be its utility with respect to A_0 and A_1. In other words, we assume that A is equivalent to the randomized alternative in which:

- we have A_1 with probability $u(A)$, and
- we have A_0 with probability $1-p$.

In the scale defined by the new alternatives A_0' and A_1', let $u'(A_0)$, $u'(A_1)$, and $u'(A)$ denote the utilities of A_0, A_1, and A. This means, in particular:

- that A_0 is equivalent to the randomized alternative in which we get A_1' with probability $u'(A_0)$ and A_0' with probability $1-u'(A_0)$; and
- that A_1 is equivalent to the randomized alternative in which we get A_1' with probability $u'(A_1)$ and A_0' with probability $1-u'(A_1)$.

Thus, the alternative A is equivalent to the compound randomized alternative, in which

- first, we select A_1 or A_0 with probabilities $u(A)$ and $1-u(A)$, and then
- depending on the first selection, we select A_1' with probability $u'(A_1)$ or $u'(A_0)$ – and A_0' with the remaining probability.

As the result of this two-stage process, we get either A_0' or A_1'. The probability p of getting A_1' in this two-stage process can be computed by using the formula of full probability

$$p = u(A) \cdot u'(A_1) + (1-u(A)) \cdot u'(A_0) = u(A) \cdot (u'(A_1) - u'(A_0)) + u'(A_0).$$

So, the alternative A is equivalent to a randomized alternative in which we get A_1' with probability p and A_0' with the remaining probability $1-p$. By definition of utility, this means that the utility $u'(A)$ of the alternative A in the scale defined by A_0' and A_1' is equal to this value p:

$$u'(A) = u(A) \cdot (u'(A_1) - u'(A_0)) + u'(A_0).$$

So, changing the scale means a linear re-scaling of the utility values:

$$u(A) \to u'(A) = \lambda \cdot u(A) + b$$

for $\lambda = u'(A_1) - u'(A_0) > 0$ and $b = u'(A_0)$.

Vice versa, for every $\lambda > 0$ and b, one can find appropriate events A_0' and A_1' for which the re-scaling has exactly these values λ and b. In other words, utility is defined modulo an arbitrary (increasing) linear transformation.

The last important aspect of the standard decision making theory is its description of the results of different actions. Suppose that an action leads to alternatives $a_1, \ldots, a_m$ with probabilities $p_1, \ldots, p_m$. We can assume that we have already determined the utility $u_i = u(a_i)$ of each of the alternatives $a_1, \ldots, a_m$. By definition of the utility, this means that for each i, the alternative a_i is equivalent to the randomized alternative $L(u_i)$ in which we get A_1 with probability u_i and A_0 with probability $1 - u_i$. Thus, the results of the action are equivalent to the two-stage process in which, with the probability p_i, we select a randomized alternative $L(u_i)$. In this two-stage process, the results are either A_1 or A_0. The probability p of getting A_1 in this two-stage process can be computed by using the formula for full probability: $p = p_1 \cdot u_1 + \ldots + p_m \cdot u_m$. Thus, the action is equivalent to a randomized alternative in which we get A_1 with probability p and A_0 with the remaining probability $1 - p$. By definition of utility, this means that the utility u of the action in question is equal to

$$u = p_1 \cdot u_1 + \ldots + p_m \cdot u_m.$$

In statistics, the right-hand of this formula is known as the *expected value*. Thus, we can conclude that the utility of each action with different possible alternatives is equal to the expected value of the utility.

5 Why Quadratic Aggregation Operations are Less Adequate than OWA and Choquet Operations: An Explanation

To adequately describe the decision maker's preferences, we must be able, given an alternative characterized by n parameters $x_1, \ldots, x_n$, to describe the utility $u(x_1, \ldots, x_n)$ of this alternative. To get a perfect description of the user's preference, we must elicit such a utility value for all possible combinations of parameters. As we have mentioned in the Introduction, for practical values n, it is not realistic to elicit that many utility values from a user. So, instead, we elicit the user's preference over each of the parameters x_i, and then aggregate the resulting utility values $u_i(x_i)$ into an approximation for $u(x_1, \ldots, x_n)$: $u(x_1, \ldots, x_n) \approx f(u_1, \ldots, u_n)$, where $u_i \stackrel{\text{def}}{=} u_i(x_i)$.

We have also mentioned that in the first approximation, linear aggregation operations $f(u_1, \ldots, u_n) = a_0 + \sum_{i=1}^{n} w_i \cdot u_i$ work well, but to get a more adequate representation of the user's preferences, we must go beyond linear functions. From the purely mathematical viewpoint, it may seem that quadratic functions $f(u_1, \ldots, u_n)$ should provide a reasonable next approximation, but in

practice, piece-wise linear aggregation operations such as OWA (or Choquet integral) provide a much more adequate description of expert preferences.

For example, for two parameters, the general OWA combination of two utility values has the form

$$f(u_1, u_2) = w_1 \cdot \min(u_1, u_2) + w_2 \cdot \max(u_1, u_2).$$

Similarly, the general OWA combination of three utility values has the form

$$f(u_1, u_2, u_3) = w_1 \cdot \min(u_1, u_2, u_3) +$$

$$w_2 \cdot \max(\min(u_1, u_2), \min(u_1, u_3), \min(u_2, u_3)) + w_3 \cdot \max(u_1, u_2, u_3).$$

Let us show that this seemingly mysterious advantage of non-quadratic aggregation operations can be explained based on the main properties of the utility functions.

Indeed, as we have mentioned in Section 2, the utility is defined modulo *two* types of transformations: changing a starting point $u \to u+b$ and changing a scale $u \to \lambda \cdot u$ for some $\lambda > 0$. It is therefore reasonable to require that the aggregation operation should not depend on which "unit" (i.e., which extreme event A_1) we use to describe utility. Let us describe this requirement in precise terms.

In the original scale,

- we start with utility values $u_1, \ldots, u_n$;
- to these values, we apply the aggregation operation $f(u_1, \ldots, u_n)$ and get the resulting overall utility $u = f(u_1, \ldots, u_n)$.

On the other hand,

- we can express the same utility values in a new scale, as $u_1' = \lambda \cdot u_1$, $\ldots$, $u_n' = \lambda \cdot u_n$;
- then, we use the same aggregation function to combine the new utility values; as a result, we get the resulting overall utility $u' = f(u_1', \ldots, u_n')$.

Substituting the expressions $u_i' = \lambda \cdot u_i$ into this formula, we conclude that $u' = f(\lambda \cdot u_1, \ldots, \lambda \cdot u_n)$. We require that the utility

$$u' = f(u_1', \ldots, u_n') = f(\lambda \cdot u_1, \ldots, \lambda \cdot u_n)$$

reflect the same degree of preference as the utility $u = f(u_1, \ldots, u_n)$ but in a different scale: $u' = \lambda \cdot u$, i.e.,

$$f(\lambda \cdot u_1, \ldots, \lambda \cdot u_n) = \lambda \cdot f(u_1, \ldots, u_n).$$

It is worth mentioning that in mathematics, such functions are called *homogeneous* (of first degree). So, we arrive at the conclusion that an adequate aggregation operation should be homogeneous.

This conclusion about the above mysterious fact. On the other hand, one can show that linear aggregation operations and piece-wise linear aggregation operations like OWA are scale-invariant.

Let us start with a linear aggregation operation

$$f(u_1, \ldots, u_n) = w_1 \cdot u_1 + \ldots + w_n \cdot u_n.$$

For this operation, we get

$$f(\lambda \cdot u_1, \ldots, \lambda \cdot u_n) = w_1 \cdot (\lambda \cdot u_1) + \ldots + w_n \cdot (\lambda \cdot u_n) =$$

$$\lambda \cdot (w_1 \cdot u_1 + \ldots + w_n \cdot u_n) = \lambda \cdot f(u_1, \ldots, u_n).$$

Let us now consider the OWA aggregation operation

$$f(u_1, \ldots, u_n) = w_1 \cdot u_{(1)} + \ldots + w_n \cdot u_{(n)},$$

where $u_{(1)}$ is the largest of n values $u_1, \ldots, u_n$, $u_{(2)}$ is the second largest, etc. If we multiply all the utility values u_i by the same constant $\lambda > 0$, their order does not change. In particular, this means that the same value $u_{(1)}$ which was the largest in the original scale is the largest in the new scale as well. Thus, its numerical value $u'_{(1)}$ can be obtained by re-scaling $u_{(1)}$: $u'_{(1)} = \lambda \cdot u_{(1)}$. Similarly, the same value $u_{(2)}$ which was the second largest in the original scale is the second largest in the new scale as well. Thus, its numerical value $u'_{(2)}$ can be obtained by re-scaling $u_{(2)}$: $u'_{(2)} = \lambda \cdot u_{(2)}$, etc. So, we have $u'_{(i)} = \lambda \cdot u_{(i)}$ for all i. Thus, for the OWA aggregation operation, we have

$$f(\lambda \cdot u_1, \ldots, \lambda \cdot u_n) = w_1 \cdot u'_{(1)} + \ldots + w_n \cdot u'_{(n)} = w_1 \cdot (\lambda \cdot u_{(1)}) + \ldots + w_n \cdot (\lambda \cdot u_{(n)}) =$$

$$\lambda \cdot (w_1 \cdot u_{(1)} + \ldots + w_n \cdot u_{(n)}) = \lambda \cdot f(u_1, \ldots, u_n).$$

On the other hand, a generic quadratic operation is not homogeneous. Indeed, a general quadratic operation has the form

$$f(u_1, \ldots, u_n) = \sum_{i=1}^{n} w_i \cdot u_i + \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot u_i \cdot u_j.$$

Here,

$$f(\lambda u_1, \ldots, \lambda u_n) = \sum_{i=1}^{n} w_i \cdot (\lambda \cdot u_i) + \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot (\lambda \cdot u_i) \cdot (\lambda \cdot u_j) =$$

$$\lambda \cdot \sum_{i=1}^{n} w_i \cdot u_i + \lambda^2 \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot u_i \cdot u_j.$$

On the other hand,

$$\lambda \cdot f(u_1, \ldots, u_n) = \lambda \cdot \sum_{i=1}^{n} w_i \cdot u_i + \lambda \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot u_i \cdot u_j.$$

The linear terms in the expressions $f(\lambda u_1, \ldots, \lambda u_n)$ and $\lambda \cdot f(u_1, \ldots, u_n)$ coincide, but the quadratic terms differ: the quadratic term in $f(\lambda u_1, \ldots, \lambda u_n)$ differs from the quadratic term in $\lambda \cdot f(u_1, \ldots, u_n)$ by a factor of λ. Thus, the only possibility to satisfy the scale-invariance (homogeneity) requirement for all λ is to have these differing quadratic terms equal to 0, i.e., to have $w_{ij} = 0$ – but in this case the aggregation operation is linear. So, quadratic operations are indeed not homogeneous – which explains whey they are less adequate in describing user's preferences than homogeneous operations like OWA or Choquet integral.

6 OWA and Choquet Operations Are, in Some Reasonable Sense, the Most General Ones: A New Result

In the previous section, we explained the empirical fact that in multi-criteria decision making, OWA and Choquet operations lead to more adequate results than seemingly natural quadratic aggregation operations. The explanation is that, due to the known properties of the utility, it is reasonable to require that aggregation operation be scale-invariant (homogeneous); OWA and Choquet operations are scale-invariant but quadratic operations are not.

However, in principle, OWA and Choquet operations are just a few examples of scale-invariant operations, so by itself, the above result does not explain why OWA and Choquet operations are so successful and not any other scale-invariant operation. In this section, we give such an explanation.

This explanation is based on the fact that OWA and Choquet operations are compositions of linear functions, min, and max. In this section, we prove that, crudely speaking, every scale-invariant operation can be composed of linear functions and min and max operations.

Definition 2. *A function $f(x_1, \ldots, x_n)$ is called* homogeneous *if for every x_1, $\ldots$, x_n and for every $\lambda > 0$, we have $f(\lambda \cdot x_1, \ldots, \lambda \cdot x_n) = \lambda \cdot f(x_1, \ldots, x_n)$.*

Definition 3. *By a* basic function*, we mean one of the following functions:*

- *a* linear function $f(x_1, \ldots, x_n) = w_1 \cdot x_1 + \ldots + w_n \cdot x_n$*;*
- *a* minimum function $f(x_1, \ldots, x_n) = \min(x_{i_1}, \ldots, x_{i_m})$*; and*
- *a* maximum function $f(x_1, \ldots, x_n) = \max(x_{i_1}, \ldots, x_{i_m})$.

We also say that basic functions are 1-level compositions *of basic functions. We say that a function $f(x_1, \ldots, x_n)$ is a* k-level composition of basic functions *if $f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_m(x_1, \ldots, x_n))$, where g is a basic function, and the functions $h_1(x_1, \ldots, x_n)$, $\ldots$, $h_m(x_1, \ldots, x_n)$ are $(k-1)$-level compositions of basic functions.*

By induction over k, one can easily prove that all compositions of basic functions are homogeneous. For example:

- a linear combination is a basic function;
- an OWA combination of two values is a 2-level composition of basic functions;
- a general OWA operation is a 3-level composition of basic functions.

It turns out that an arbitrary homogeneous function can be approximated by appropriate 3-level compositions.

Definition 4. *Let $k > 0$ be a positive integer. We say that k-level compositions have a* universal approximation property for homogeneous functions *if for every continuous homogeneous function $f(x_1, \ldots, x_n)$, and for every two numbers $\varepsilon > 0$ and $\Delta > 0$, there exists a function $\widetilde{f}(x_1, \ldots, x_n)$ which is a k-level composition of basic functions and for which*

$$|f(x_1, \ldots, x_n) - \widetilde{f}(x_1, \ldots, x_n)| \le \varepsilon$$

for all $x_1, \ldots, x_n$ for which $|x_i| \le \Delta$ for all i.

Theorem 2. *3-level compositions have a universal approximation property for homogeneous functions.*

A natural question is: do we need that many levels of composition? What is we only use 1- or 2-level compositions? It turns out that in this case, we will not get the universal approximation property – and thus, the 3 levels of OWA operations is the smallest possible number.

Theorem 3.

- *1-layer computations do not have a universal approximation property for homogeneous functions;*
- *2-layer computations do not have a universal approximation property for homogeneous functions.*

Comment. A natural question is: why should we select linear functions, min, and max as basic functions? One possible answer is that these operations are the fastest to compute, i.e., they require the smallest possible number of computational steps.

Indeed, the fastest computer operations are the ones which are hardware supported, i.e., the ones for which the hardware has been optimized. In modern computers, the hardware supported operations with numbers include elementary arithmetic operations ($+$, $-$, $\cdot$, $/$, etc.), and operations min and max.

In the standard (digital) computer (see, e.g., [3]):

- addition of two n-bit numbers requires, in the worst case, $2n$ bit operations: n to add corresponding digits, and n to add carries;
- multiplication, in the worst case, means n additions – by each bit of the second factor; so, we need $O(n^2)$ bit operations;

- division is usually performed by trying several multiplications, so it takes even longer than multiplication;
- finally, min and max can be performed bit-wise and thus, require only n bit operations.

Thus, the fastest elementary operations are indeed addition (or, more generally, linear combination), min, and max.

Proof of Theorems 2 and 3.

1°. Before we start proving, let us notice that the values of the functions $\min(x_{i_1}, \ldots, x_{i_m})$ and $\max(x_{i_1}, \ldots, x_{i_m})$ depend on the order between the values $x_1, \ldots, x_n$. There are $n!$ possible orders, so we can divide the whole n-dimensional space of all possible tuples $(x_1, \ldots, x_n)$ into $n!$ zones corresponding to these different orders.

2°. In each zone, a basic function is linear:

- a linear function is, of course, linear;
- a minimizing function $\min(x_{i_1}, \ldots, x_{i_m})$ is simply equal to the variable x_{i_k} which is the smallest in this zone and is, thus, linear;
- a maximizing function $\max(x_{i_1}, \ldots, x_{i_m})$ is simply equal to the variable x_{i_k} which is the largest in this zone and is, thus, also linear.

3°. If a function $f(x_1, \ldots, x_n)$ can be approximated, with arbitrary accuracy, by functions from a certain class, this means that $f(x_1, \ldots, x_n)$ is a limit of functions from this class.

4°. Basic functions are linear in each zone; thus, their limits are also linear in each zone. Since some homogeneous functions are non-linear, we can thus conclude that basic functions do not have a universal approximation property for homogeneous functions.

5°. Let us now consider 2-level compositions of basic functions, i.e., functions of the type $f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_m(x_1, \ldots, x_n))$, where g and h_i are basic functions.

Since there are three types of basic functions, we have three options:

- it is possible that $g(x_1, \ldots, x_m)$ is a linear function;
- it is possible that $g(x_1, \ldots, x_m)$ is a minimizing function; and
- it is possible that $g(x_1, \ldots, x_m)$ is a maximizing function.

Let us consider these three options one by one.

5.1°. Let us start with the first option, when $g(x_1, \ldots, x_m)$ is a linear function. Since on each zone, each basic function h_i is also linear, the composition $f(x_1, \ldots, x_n)$ is linear on each zone.

5.2°. If $g(x_1, \ldots, x_m)$ is a minimizing function, then on each zone, each h_i is linear and thus, the composition $f(x_1, \ldots, x_n)$ is a minimum of linear functions. It is known that minima of linear functions are concave; see, e.g., [21]. So, within this option, the function $f(x_1, \ldots, x_n)$ is concave.

5.3°. If $g(x_1, \ldots, x_m)$ is a maximizing function, then on each zone, each h_i is linear and thus, the composition $f(x_1, \ldots, x_n)$ is a maximum of linear functions. It is known that maxima of linear functions are convex; see, e.g., [21]. So, within this option, the function $f(x_1, \ldots, x_n)$ is convex.

6°. In each zone, 2-level compositions of basic functions are linear, concave, or convex. The class of all functions approximable by such 2-level compositions is the class of limits (closure) of the union of the corresponding three classes: of linear, concave, and convex sets. It is known that the closure of the finite union is the union of the corresponding closures. A limit of linear functions is always linear, a limit of concave functions is concave, and a limit of convex functions is convex. Thus, by using 2-level compositions, we can only approximate linear, concave, or convex functions. Since there exist homogeneous functions which are neither linear nor concave or convex, we can thus conclude that 2-level compositions are not universal approximators for homogeneous functions.

7°. To complete the proof, we must show that 3-level compositions are universal approximators for homogeneous functions. There are two ways to prove it.

7.1°. First, we can use the known facts about concave and convex functions [21]:

- that every continuous function on a bounded area can be represented as as a difference between two convex functions, and
- that every convex function can be represented as a maximum of linear functions – namely, all the linear functions which are smaller than this function.

These facts are true for general (not necessarily homogeneous) functions. For homogeneous functions $f(x_1, \ldots, x_n)$, one can easily modify the existing proofs and show:

- that every homogeneous continuous function on a bounded area can be represented as as a difference between two convex homogeneous functions, and
- that every homogeneous convex function can be represented as a maximum of homogeneous linear functions – namely, all the homogeneous linear functions which are smaller than this function.

Thus, we can represent the desired function $f(x_1, \ldots, x_n)$ as the difference between two convex homogeneous functions $f(x_1, \ldots, x_n) = f_1(x_1, \ldots, x_n) - f_2(x_1, \ldots, x_n)$. Each of these convex functions can be approximated by maxima of linear functions and thus, by 2-level compositions. Substraction $f_1 - f_2$ adds the third level, so $f(x_1, \ldots, x_n)$ can indeed be approximated by 3-level compositions.

To prove that a function $f(x_1, \ldots, x_n)$ can be represented as a different between two convex functions, we can, e.g., first approximate it by a homogeneous function which is smooth on a unit sphere

$$\{(x_1, \ldots, x_n) : x_1^2 + \ldots + x_n^2 = 1\},$$

and then take $f_1(x_1, \ldots, x_n) = k \cdot \sqrt{x_1^2 + \ldots + x_n^2}$ for a large k. For smooth functions, convexity means that the Hessian matrix – consisting of its second derivatives $\dfrac{\partial^2 f}{\partial x_i \partial x_j}$ – is positive definite.

For sufficiently large k, the difference

$$f_2(x_1, \ldots, x_n) = f_1(x_1, \ldots, x_n) - f(x_1, \ldots, x_n)$$

is also convex – since its second derivatives matrix is dominated by positive definite terms coming from f_1. Thus, the difference $f_1 - f_2 = f$ is indeed the desired difference.

7.2°. Another, more constructive proof, is, for some $\delta' > 0$, to select a finite δ'-dense set of points $e = (e_1, \ldots, e_n)$ on a unit square. For each such point, we build a 2-level composition which coincides with f on the corresponding ray $\{\lambda \cdot (e_1, \ldots, e_n) : \lambda > 0\}$. This function can be obtained, e.g., as a minimum of several linear functions which have the right value on this ray but change drastically immediately outside this ray.

For example, let $f_0(x)$ be an arbitrary homogeneous linear function which coincides with $f(x)$ at the point e – and thus, on the whole ray. To construct the corresponding linear functions, we can expand the vector e to an orthonormal basis e, e', e'', etc., and take linear functions $f_0(x) + k \cdot (e' \cdot x)$ and $f_0(x) - k \cdot (e' \cdot x)$ for all such e' (and for a large $k > 0$). Then, the minimum of all these functions is very small outside the ray.

We then take the maximum of all these minima – a 3-level composition.

The function $f(x_1, \ldots, x_n)$ is continuous on a unit sphere and thus, uniformly continuous on it, i.e., for every $\varepsilon > 0$, there is a δ such that δ-close value on the unit sphere lead to ε-close values of f. By selecting appropriate δ' and k (depending on δ), we can show that the resulting maximum is indeed ε-close to f.

The theorem is proven.

7 Conclusions

Fuzzy techniques have been originally invented as a methodology that transforms the knowledge of experts (formulated in terms of natural language) into a precise computer-implementable form. There are many successful applications of this methodology to situations in which expert knowledge exist; the most well known (and most successful) are applications to fuzzy control.

In some cases, fuzzy methodology is applied even when no expert knowledge exists. In such cases, instead of trying to approximate the unknown control function by splines, polynomials, or by any other traditional approximation technique, researchers try to approximate it by guessing and tuning the expert rules. Surprisingly, this approximation often works fine.

Similarly, in multi-criteria decision making, it is necessary to aggregate (combine) utility values corresponding to several criteria (parameters). The simplest way to combine these values is to use linear aggregation. In many practical situations, however, linear aggregation does not fully adequately describe the actual decision making process, so non-linear aggregation is needed. From the purely mathematical viewpoint, the next natural step after linear functions is the use of quadratic functions. However, in decision making, a different type of non-linearities are usually more adequate than quadratic ones: fuzzy-type non-linearities like OWA or Choquet integral that use min and max in addition to linear combinations.

In this paper, we give a mathematical explanation for this empirical phenomenon. Specifically, we show that approximation by using fuzzy methodology is indeed the best (in some reasonable sense).

Acknowledgments

This work was supported in part by NSF grants HRD-0734825, EAR-0225670, and EIA-0080940, by Texas Department of Transportation grant No. 0-5453, by the Japan Advanced Institute of Science and Technology (JAIST) International Joint Research Grant 2006-08, and by the Max Planck Institut für Mathematik.

References

1. J. J. Buckley, "Sugeno type controllers are universal controllers", *Fuzzy Sets and Systems*, 1993, Vol. 53, pp. 299–303.
2. M. Ceberio and F. Modave, "An interval-valued, 2-additive Choquet integral for multi-cruteria decision making", *Proceedings of the 10th Conf. on Information Processing and Management of Uncertainty in Knowledge-based Systems IPMU'04*, Perugia, Italy, July 2004.
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 2001.
4. R. Feynman, R. Leighton, and M. Sands, *Feynman Lectures on Physics*, Addison Wesley, 2005.
5. M. Grabisch, T. Murofushi, and M. Sugeno (eds.), *Fuzzy Measures and Integrals*, Physica-Verlag, Berlin-Heidelberg, 2000.
6. A. Kandel and G. Langholtz (eds.), *Fuzzy Control Systems*, CRC Press, Boca Raton, FL, 1994.
7. R. L. Keeney and H. Raiffa, *Decisions with Multiple Objectives*, John Wiley and Sons, New York, 1976.
8. G. Klir and B. Yuan, *Fuzzy sets and fuzzy logic: theory and applications*, Prentice Hall, Upper Saddle River, NJ, 1995.
9. B. Kosko, "Fuzzy systems as universal approximators", *Proceedings of the 1st IEEE International Conference on Fuzzy Systems*, San Diego, CA, 1992, pp. 1153–1162.

10. V. Kreinovich and A. Bernat, "Parallel algorithms for interval computations: an introduction", *Interval Computations*, 1994, No. 3, pp. 6–62.
11. V. Kreinovich, G. C. Mouzouris, and H. T. Nguyen, "Fuzzy rule based modeling as a universal approximation tool", In: H. T. Nguyen and M. Sugeno (eds.), *Fuzzy Systems: Modeling and Control*, Kluwer, Boston, MA, 1998, pp. 135–195.
12. V. Kreinovich, H. T. Nguyen, and Y. Yam, "Fuzzy systems are universal approximators for a smooth function and its derivatives", *International Journal of Intelligent Systems*, 2000, Vol. 15, No. 6, pp. 565–574.
13. R. N. Lea and V. Kreinovich, "Intelligent control makes sense even without expert knowledge: an explanation", *Reliable Computing*, 1995, Supplement (Extended Abstracts of APIC'95: International Workshop on Applications of Interval Computations, El Paso, TX, Febr. 23–25, 1995), pp. 140–145.
14. R. D. Luce and H. Raiffa, *Games and Decisions: Introduction and Critical Survey*, Dover, New York, 1989.
15. F. Modave, M. Ceberio, and V. Kreinovich, "Choquet integrals and OWA criteria as a natural (and optimal) next step after linear aggregation: a new general justification", *Proceedings of the 7th Mexican International Conference on Artificial Intelligence MICAI'08*, Mexico City, Mexico, October 27–31, 2008, to appear.
16. H. T. Nguyen and V. Kreinovich, "On approximation of controls by fuzzy systems", *Proceedings of the Fifth International Fuzzy Systems Association World Congress*, Seoul, Korea, July 1993, pp. 1414–1417.
17. H. T. Nguyen and V. Kreinovich, "Fuzzy aggregation techniques in situations without experts: towards a new justification", *Proceedings of the IEEE Conference on Foundations of Computational Intelligence FOCI'07*, Hawaii, April 1–5, 2007, pp. 440–446.
18. H. T. Nguyen and E. A. Walker, *A first course in fuzzy logic*, CRC Press, Boca Raton, Florida, 2005.
19. I. Perfilieva and V. Kreinovich, "A new universal approximation result for fuzzy systems, which reflects CNF-DNF duality", *International Journal of Intelligent Systems*, 2002, Vol. 17. No. 12, pp. 1121–1130.
20. H. Raiffa, *Decision Analysis*, Addison-Wesley, Reading, Massachusetts, 1970.
21. R. T. Rockafeller, *Convex Analysis*, Princeton University Press, Princeton, New Jersey, 1970.
22. L.-X. Wang, "Fuzzy systems are universal approximators", *Proceedings of the IEEE International Conference on Fuzzy Systems,* San Diego, CA, 1992, pp. 1163–1169.
23. L.-X. Wang and J. Mendel, *Generating Fuzzy Rules from Numerical Data, with Applications*, University of Southern California, Signal and Image Processing Institute, Technical Report USC-SIPI # 169, 1991.
24. R. R. Yager and J. Kacprzyk (eds.), *The Ordered Weighted Averaging Operators: Theory and Applications*, Kluwer, Norwell, Massachusetts, 1997.
25. R. R. Yager and V. Kreinovich, "Universal approximation theorem for uninorm-based fuzzy systems modeling", *Fuzzy Sets and Systems*, 2003, Vol. 140, No. 2, pp. 331–339.