University of Texas at El Paso
ScholarWorks@UTEP

Departmental Technical Reports (CS) | Computer Science

10-1-2024

Two Is Enough, but Three (or More) Is Better: in AI and Beyond

Olga Kosheleva
University of Texas at El Paso, olgak@utep.edu

Vladik Kreinovich
University of Texas at El Paso, vladik@utep.edu

Victor L. Timchenko
Admiral Makarov National Shipbuilding University, vl.timchenko58@gmail.com

Yury P. Kondratenko
Petro Mohyla Black Sea National University, y_kondrat2002@yahoo.com

Follow this and additional works at: https://scholarworks.utep.edu/cs_techrep

 Part of the Computer Sciences Commons, and the Mathematics Commons

Comments:
Technical Report: UTEP-CS-24-51

Recommended Citation

Kosheleva, Olga; Kreinovich, Vladik; Timchenko, Victor L.; and Kondratenko, Yury P., "Two Is Enough, but Three (or More) Is Better: in AI and Beyond" (2024). *Departmental Technical Reports (CS)*. 1907.
https://scholarworks.utep.edu/cs_techrep/1907

This Article is brought to you for free and open access by the Computer Science at ScholarWorks@UTEP. It has been accepted for inclusion in Departmental Technical Reports (CS) by an authorized administrator of ScholarWorks@UTEP. For more information, please contact lweber@utep.edu.

Two Is Enough, but Three (or More) Is Better: in AI and Beyond

Olga Kosheleva[1], Vladik Kreinovich[1],
Victor Timchenko[2], and Yuriy Kondratenko[3]
[1]University of Texas at El Paso, El Paso, Texas, USA
olgak@utep.edu, vladik@utep.edu
[2]Admiral Makarov National University of Shipbuilding,
Mykolaiv, Ukraine, vl.timchenko58@gmail.com
[3]Black Sea State University named after Petro Mohyla,
Mykolaiv, Ukraine, and
Institute of Artificial Intelligence Problems
of MES and NAS of Ukraine, Kyiv, Ukraine,
y_kondrat2002@yahoo.com

Abstract

At present, the most successful AI technique is deep learning – the use of neural networks that consist of multiple layers. Interestingly, it is well known that neural networks with two data processing layers are sufficient – in the sense that they can approximate any function with any given accuracy. Because of this, until reasonably recently, researchers and practitioners used such networks. However, recently it turned out, somewhat unexpectedly, that using three or more data processing layers – i.e., using what is called deep learning – makes the neural networks much more efficient. In this paper, on numerous examples from AI and from beyond AI, we show that this is a general phenomenon: two is enough but three or more is better. In many examples, there is a specific explanation for this phenomenon. However, the fact that this phenomenon is universal makes us conjecture that there is a general explanation for this phenomenon – and we provide a possible explanation.

1 Deep learning vs. traditional shallow neural networks

What is a neural network: a brief reminder. Artificial neural networks consists of *neurons*, computational units that crudely emulate biological neurons in our brain. A neuron takes several inputs $s_1, \ldots, s_m$, and outputs the value $s = a(w_0 + w_1 \cdot s_1 + \ldots w_m \cdot s_m)$, where:

- w_i are constants called *weights* and
- $a(z)$ is a function called *activation function*;

see, e.g., [3, 9].

In a neural network for processing data $x_1, \ldots, x_n$:

- some neurons directly process the data – these neurons form the first data processing layer, while
- other neurons process the results of the previous layer(s).

Neural networks with two data processing layers are universal approximators. It is well known that neural networks with two data processing layers are universal approximators – in the sense that:

- for each continuous function $f(x_1, \ldots, x_n)$ on a bounded domain D and for every $\varepsilon > 0$,
- there exists a neural network that, for each $(x_1, \ldots, x_n) \in D$, produces a value which is ε-close to $f(x_1, \ldots, x_n)$;

see, e.g., [3, 9, 14].

Because of this, until reasonably recently, researchers and practitioners used such networks.

Enter deep learning. Recently, it turned out, somewhat unexpectedly, that neural networks with three or more data processing layers – called *deep* neural networks – are much more efficient [9]. Everyone knows spectacular successes of deep learning.

How can we explain this success? A possible explanation for this phenomenon was proposed in [1]. This explanation is based on the need for the most efficient universal approximation.

While from the purely mathematical viewpoint, the weights can be any real numbers, in practice, we need to store these numbers in a computer, and we have a limited number of bits that we can store. We have a limited number of bits available – let us denote this number by B. So, we can have no more than 2^B different combinations of bits – and thus, we can represent no more than 2^B different functions.

The more different functions we can represent, the more accurate is the resulting approximation. For example:

- If we can only use one number to approximate all the numbers from the interval $[0, 1]$, then the best we can do is select the number 0.5 – that approximates all numbers from this interval with accuracy $[0, 1]$.
- If we are allowed to use two numbers, then we can use $1/4$ and $3/4$ and get the approximation accuracy 0.25, etc.

If we have K neurons in a layer, then each of $K!$ permutations of these K neurons creates a different binary sequence – but all these binary sequences represent the same function. Thus, instead of 2^B possible functions, we can only represent $2^B/K!$ different function. To represent more functions – and thus, to make approximations more accurate – we need to decrease K, i.e., decrease the number of neurons in each layer. So, instead of placing all data processing neurons in a single layer, a better idea is to have several layers – which is exactly what deep learning does.

2 Why neural networks in the first place?

Functions of two variables are sufficient. It is know that any continuous function can be represented as a superposition of functions of two variables. This well-known result by the famous Russian mathematician Andrei Kolmogorov solved one of the 23 challenges that the 19 century mathematicians, under the leadership of David Hilbert, provided to the 20 century mathematics [13].

In line with this result, most modern computers directly support functions of one or two variables – such as min, max, addition, subtraction, multiplication, division, etc. – and all other functions have to be computed as a composition of these operations.

However, supporting functions of many variables makes computations more efficient. The relative efficiency of supporting operations with more than two inputs can be illustrated by the fact that most computer chips now support a three-input *multiply-and-add* operation $a + b \cdot c$.

However, the most impressive jump in efficiency can be observed when we switch to multi-variable neural operations.

How can we explain this efficiency. A possible explanation was given in [8, 15]. One of the main objectives of a computer is to compute as fast as possible. We can speed up computations by performing several operations in parallel – and nowadays, even the cheapest computers have several processors working in parallel. Parallelization reduced the overall time of several parallelized elementary operation to the time for performing one of them. So, to make parallelization as efficient as possible, we need to make these elementary operations as fast as possible.

In general, the fastest-to-compute are linear functions. However, we cannot limit ourselves only to linear functions. Indeed:

- by combining linear functions, we will only get linear functions, and
- many real-life phenomena are nonlinear.

So, in addition to linear functions, we also need to use some nonlinear ones. In general, the more inputs a nonlinear function has, the longer it takes to compute this function. Thus, if we want to minimize computation time, we should limit ourselves to fastest-to-compute nonlinear functions, i.e., to nonlinear function of one variable.

In a sequence of computations, it makes no sense to follow a linear operation with a linear one, since the resulting composition of linear function is also linear and thus, can be computed in a single linear stage. Similarly, it makes no sense to follow a nonlinear operation with another nonlinear operation, since the resulting composition of two functions of one variable is also a function of one variable and thus, can be computed in a single nonlinear stage. Thus, a linear stage – of computing a linear combination $z = w_0 + w_1 \cdot \cdots s_1 + \ldots + w_m \cdot s_m$ – should always be followed by a stage at which a function $a(z)$ is applied to this linear combination. The result is exactly what the usual artificial neuron computes – which explains why neural networks are so efficient in the first place.

3 From binary logic to multiple-valued logics

Traditional two-valued logic is sufficient. Everything can be represented in a computer, and everything in a computer is represented as a sequence of 0s and 1s. So, if we use the usual interpretation of 1 as true and 0 as false, we can represent all our knowledge by using a logic that uses only two values: true and false.

Because of this, until 1960s, most computer applications used the two-valued logic.

However, multiple-valued logics are often more efficient. Starting with 1960s, it was discovered that multiple-valued logics such as fuzzy logic (see, e.g., [2, 11, 19, 20, 21, 33]) often lead to successful practical applications.

In particular, we can make computations more efficient if we represent different truth values as combinations of optical beams of three basic colors; see, e.g., [23, 24, 25, 26, 27, 28, 29, 30, 31, 32].

How this can be explained. Knuth's fundamental book on foundations of computing [12] contains a result that while binary representation are sufficient, multiple-valued (e.g., ternary) representations are more efficient.

4 From traditional fuzzy logic to higher-order fuzzy logic

Traditional fuzzy logic: a brief reminder. Fuzzy logic was invented by Lotfi Zadeh to describe imprecise ("fuzzy") expert statements like "x is small" in precise computer-understandable terms. For this purpose, Zadeh suggested to ask an expert:

- for each possible value x of the corresponding quantity,
- to mark, on the scale from 0 to 1, a degree which this value is consistent with the statement – e.g., to which this value is small.

The resulting function that assigns, to each possible value x, the corresponding degree $\mu(x)$ is known as a *membership function*, or, alternatively, a *fuzzy set*.

Need for higher-order fuzzy logic. The very need for fuzzy logic comes from the fact that the expert is often unable to provide an exact estimate for a quantity. Instead, he/she provides a fuzzy estimate like "x is close to 1". However:

- just like the expert cannot provide the exact value of the physical quantity,
- the same expert is similarly unable to describe his/her degree of confidence by a precise number.

What the expert *can* do is describe this degree by a fuzzy word. If we use Zadeh's idea to transform this word into a fuzzy set, then:

- instead of assigning, to each value x, a degree $\mu(x)$,
- we assign, to each value x, a fuzzy set.

This construction is known as *type-2 fuzzy logic*; see, e.g., [19].

In a type-2 fuzzy set, we assign:

- to each value x and to each possible fuzzy truth value d,
- a degree $\mu(x, d)$ to which d adequately describes the expert's opinion.

Comment. Of course, instead of providing the exact value $\mu(x, d)$, the expert may also provide an imprecise estimate – and describing this estimate as a fuzzy set leads to type-3 fuzzy logic, etc.

In principle, type-2 is sufficient. It can proven (see, e.g., [18]) that from the purely mathematical viewpoint, type-2 description is sufficient. Indeed, for each possible value x, the expert provides a word w describing his/her degree that x is the actual value. The expert's meaning of this word is fully described by asking the expert to provide, for each possible degree d, a word a describing to what extent d is an adequate description of w. In other words, a full description of this degree is provided by a function of one variable.

A type-1 single value $\mu(x)$ is not sufficient to uniquely determine such a function. However, in type-2, we already have such a function – namely, the function $d \mapsto \mu(x, d)$. So, we have enough information to determine the desired function $s \mapsto a$. (Of course, what we just described is just an idea; see [18] for the full proof.)

Because of this and similar results, most higher-order fuzzy techniques limit themselves to type-2.

However, sometimes, type-3 fuzzy techniques are more efficient. However, interestingly, in some cases, type-3 techniques turn out to be more efficient, see, e.g., [4, 5, 6, 7, 17].

5 Computability

Which computational devices are sufficient: a brief reminder. Before we start thinking of an algorithm for solving a general problem, it is important to first figure out whether this problem is algorithmically solvable in the first place. Some problems are not algorithmically solvable. For example, it is not possible to always predict whether a given computer program will halt on a given data. For such problems, it is desirable to avoid wasting time trying to find a general solution.

Analysis of computability is one of the main subjects of theory of computation. Courses on theory of computing (see, e.g., [22]) usually start with the simplest possible computational device – a finite automaton. For this device, we have a finite number of possible states $q_1, \ldots, q_n$. One of these states is marked as a starting state, and a subset of the states are marked as final states. We fix a finite set of possible symbols $s_1, \ldots, s_m$.

To complete the description of a finite automaton, we need to describe, for each state q_i and for each symbol s_j read by the automaton, what will be the next state. Then, to check whether a word (i.e., a sequence of symbols) is accepted by this automaton, we:

- start in the starting state, and
- read the symbols from this word one by one – and each time move to a new state in accordance with the automaton's description.

If at the end, after reading all the symbols from the given word, we end up in a final state, the word is accepted; otherwise, the word is not accepted.

Many properties can be detected by finite automata, but it is proven that finite automata are not sufficient. For example, finite automata cannot detect words of the type $a^n b^n$ in which:

- first a symbol a is repeated some number of times, and
- then another symbol b is repeated the exact same number of times.

This detection is important for computing, since, e.g., we need to make sure that the number of opening curly brackets is exactly the same as number of closing curly brackets.

The main reason why finite automata cannot detect such words that is that they do not have memory. So, a natural idea is to provide them with some data type that would enable us to store information. There are many possible data types – lists, queues, etc., but the most natural to a computer (and thus the fastest to implement) is stack, last-come-first-served, since stack is how computer memory is operated.

In a stack, we have two main operations:

- *push*, when we place a new element on top of the stack, and
- *pop*, when we delete the top element from the stack.

So, the natural next computational device is an finite automaton equipped with a stack; this combination is called a *pushdown automaton*. For pushdown automata, for each state q_i and for each read symbol s_j:

- we can pop the top symbol from the stack, and,
- depending on which symbol pops out, go to a different state and/or push another symbol into the stack.

We start at the starting state with an empty stack. If we end up in the final state with an empty stack, the word is accepted.

Pushdown automata can detect many properties, but some properties cannot be detected by them. For example, pushdown automata cannot detect words of the type $a^n b^n c^n$ in which:

- first a symbol a is repeated some number of times,
- then another symbol b is repeated the exact same number of times, and
- then yet another symbol c is repeated the same number of time.

Textbooks usually explain that we need a *Turing machine* to recognize such properties. A Turing machine consists of:

- a tape consisting of cells and
- a head – which is a finite automaton.

The tape is infinite in one direction: it has a starting state, and potentially infinitely many following states. Initially, the head points at a starting state. The input word is written in the tape. Upon reading a symbol, depending on what symbol it sees and on what is the head's state, the head can do some (or all) of the following three things:

- it can change its state,
- it can overwrite the symbol in the cell to which it points, and/or
- it can move one step to the left or one step to the right.

Again, two are sufficient. What the textbooks usually do not mention is that a Turing machine can be equivalently represented as a finite automaton with two stacks [16]:

- the first stack contains all the symbols on the tape from the head location to the left, with the symbol to which the head points at the top, while
- the second stack contains all the other symbols, which the symbol to the direct right of the head at the top.

One can show that one step of a Turing machine can be described in terms of pushing and popping with these two stacks.

Two stacks are sufficient, because Turing machine can compute anything that can be done by a finite automaton with many stacks.

Again, more than two are sometimes more efficient. There are many problems for which a Turing machine with several tapes – corresponding to more than more than stacks – is more efficient that the basic Turing machine with a single tape (that corresponds to two stacks).

6 Is there a general explanation for this phenomenon?

There probably is a general explanation. In this paper, we provided several examples of the phenomenon. In many cases, we have a specific explanation. However, the very fact that this phenomenon is universal makes us conjecture that there is a general explanation for this phenomenon.

So what is a possible explanation? To come up with such an explanation, let us look into the most fundamental level of the physical world – the level of quantum physics. On this level, there is a following fundamental result [10]: that algebras used in quantum physics can be determined by their lowest 2 levels:

- level of "atoms" (i.e., basic building blocks), and
- level of combinations of two "atoms".

This is exactly what we observe – that in many cases, two levels are sufficient. So this is probably the general explanation for all the above phenomena.

Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science), HRD-1834620 and HRD-2034030 (CAHSI Includes), EAR-2225395 (Center for Collective Impact in Earthquake Science C-CIES), and by the AT&T Fellowship in Information Technology.

It was also supported by a grant from the Hungarian National Research, Development and Innovation Office (NRDI), and by the Institute for Risk and Reliability, Leibniz Universitaet Hannover, Germany.

References

[1] C. Baral, O. Fuentes, and Vladik Kreinovich, "Why Deep Neural Networks: A Possible Theoretical Explanation", In: M. Ceberio and V. Kreinovich

(eds.), *Constraint Programming and Decision Making: Theory and Applications*, Springer Verlag, Berlin, Heidelberg, 2018, pp. 1–6.

[2] R. Belohlavek, J. W. Dauben, and G. J. Klir, *Fuzzy Logic and Mathematics: A Historical Perspective*, Oxford University Press, New York, 2017.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2006.

[4] O. Castillo, J. Castro, and P. Melin, *Interval Type-3 Fuzzy Systems: Theory and Design*, Springer, Cham, Switzerland, 2022.

[5] O. Castillo, J. Castro, and P. Melin, "A methodology for building interval type-3 fuzzy systems based on the principle of justifiable granularity", *International Journal of Intelligent Systems*, 2022, doi 10.1002/int.22910

[6] O. Castillo, J. Castro, and P. Melin, "Interval type-3 fuzzy aggregation of neural networks for multiple time series prediction: the case of financial forecasting", *Axioms*, 2022, Vol. 11, Paper 251.

[7] O. Castillo, J. Castro, and P. Melin, "Interval type-3 fuzzy control for automated tuning of image quality in televisions", *Axioms*, 2022, Vol. 11, Paper 276.

[8] J. Contreras, M. Ceberio, O. Kosheleva, and V. Kreinovich, "Why neural networks in the first place: a theoretical explanation", *Journal of Intelligent and Fuzzy Systems*, 2022, Vol. 43, No. 6, pp. 6947–6951.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, Massachusetts, 2016.

[10] J. Harding, C. Heunen, B. Lindenhovious, and M. Navara, "Boolean subalgebras of orthoalgebras", *Order*, 2019, Vol. 36, No. 3, pp. 567–609.

[11] G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.

[12] D. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd edition, Addison Wesley, Boston, Massachusetts, 1998.

[13] A. N. Kolmogorov, "On the Representation of Continuous Functions of Several Variables by Superposition of Continuous Functions of One Variable and Addition", *Dokl. Akad. Nauk SSSR*, 1957, Vol. 114, pp. 369–373.

[14] V. Kreinovich, "Arbitrary nonlinearity is sufficient to represent all functions by neural networks: a theorem," *Neural Networks*, 1991, Vol. 4, 381–383.

[15] V. Kreinovich and A. Bernat, "Parallel algorithms for interval computations: an introduction", *Interval Computations*, 1994, No. 3, pp. 6–62.

[16] V. Kreinovich and O. Kosheleva, "A Turing Machine Is Just a Finite Automaton with Two Stacks: A Comment on Teaching Theory of Computation", *Proceedings of the 8th International Scientific-Practical Conference "Mathematical Education in Schools and Universities: Innovations in the Information Space" MATHEDU'2018*, Kazan, Russia, October 17–21, 2018, pp. 152–156.

[17] V. Kreinovich, O. Kosheleva, P. Melin, and O. Castillo, "Efficient algorithms for data processing under type-3 (and higher) fuzzy uncertainty", *Mathematics*, 2022, Vol. 10, Paper 2361.

[18] V. Kreinovich and H. T. Nguyen, "1st Order, 2nd Order, What Next? Do We Really Need Third-Order Descriptions: A View From A Realistic (Granular) Viewpoint", *Proceedings of the Joint 9th World Congress of the International Fuzzy Systems Association and 20th International Conference of the North American Fuzzy Information Processing Society IFSA/NAFIPS 2001*, Vancouver, Canada, July 25–28, 2001, pp. 1908–1913.

[19] J. M. Mendel, *Explainable Uncertain Rule-Based Fuzzy Systems*, Springer, Cham, Switzerland, 2024.

[20] H. T. Nguyen, C. L. Walker, and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2019.

[21] V. Novák, I. Perfilieva, and J. Močkoř, *Mathematical Principles of Fuzzy Logic*, Kluwer, Boston, Dordrecht, 1999.

[22] M. Sipser, *Introduction to the Theory of Computation*, 3rd edition, Course Technology Inc., Boston, Massacusetts, USA, 2021.

[23] V. Timchenko, Y. Kondratenko, O. Kozlov, and V. Kreinovich, "Fuzzy Color Computing Based on Optical Logical Architecture", In: C. Kahraman, I. U. Sari, B. Oztaysi, S. Cebi, S. C. Onar, and A. C. Tolga (eds.), *Intelligent and Fuzzy Systems: Intelligence and Sustainable Future, Proceedings of the International Conference on Intelligent and Fuzzy Systems INFUS'2023, Istanbul, Turkey, August 22–24, 2023, Vol. 1*, Springer Lecture Notes in Networks and Systems, 2023, Vol. 758, pp. 491–499, https://doi.org/10.1007/978-3-031-39774-5_55

[24] V. L. Timchenko, Y. P. Kondratenko, and V. Kreinovich, "Why Color Optical Computing", In: N. H. Phuong and V. Kreinovich (eds.), *Deep Learning and Other Soft Computing Techniques: Biomedical and Related Applications*, Studies in Computational Intelligence, Vol. 1097, Springer, Cham, Switzerland, 2023, pp. 227–233, https://doi.org/10.1007/978-3-031-29447-1_20

[25] V. L. Timchenko, Y. P. Kondratenko, and V. Kreinovich, "Logical Decision Networks Based on the Optical Coloroids", *Proceedings of the 12th IEEE International Conference on Intelligent Data Acquisition and*

Advanced Computing Systems: Technology and Applications IDAACS 2023, Dortmund, Germany, September 7–9, 2023, pp. 1194–1199, DOI: 10.1109/IDAACS58523.2023.10348692.

[26] V. L. Timchenko, Y. P. Kondratenko, and V. Kreinovich, "Interval-Valued and Set-Valued Extensions of Discrete Fuzzy Logics, Belnap Logic, and Color Optical Computing", In: Sebastia Massanet, Susana Montes, Daniel Ruiz-Aguilera, and Manuel González-Hidalgo (Eds.), *Fuzzy Logic and Technology, and Aggregation Operators. Proceedings of the 13th Conference of the European Society for Fuzzy Logic and Technology EUSFLAT 2023 and 12th International Summer School on Aggregation Operators AGOP 2023, Palma de Mallorca, Spain, September 4–8, 2023*, Springer Lecture Notes in Computer Science, 2023, Vol. 14069, pp. 297–303, doi https://doi.org/10.1007/978-3-031-39965-7_25

[27] V. L. Timchenko, Y. P. Kondratenko, and V. Kreinovich, "The architecture of optical logical coloroid with fuzzy computing", *Proceedings of the 4th International Workshop on Intelligent Information Technologies & Systems of Information Security IntelITSIS'2023*, Khmelnitsky, Ukraine, March 22–24, 2023, CEUR Workshop Proceedings, 2023, Vol. 3373, pp. 638–648.

[28] V. L. Timchenko, Y. P. Kondratenko, and V. Kreinovich, "Implementation of Optical Logic Gates Based on Color Filters", In: Z. Hu, I. Dychka, and M. He (Eds.), *Advances in Computer Science for Engineering and Education VI. Proceedings of the 6th International Conference on Computer Science, Engineering and Education Applications ICCSEEA 2023, Warsaw, Poland, March 17–19, 2023*, Springer Lecture Notes on Data Engineering and Communications Technologies, 2023, Vol. 181, pp. 126–136. https://doi.org/10.1007/978-3-031-36118-0_12

[29] V. L. Timchenko, Y. P. Kondratenko, and V. Kreinovich, "Logical Platforms for Mobile Application in Decision Support Systems Based on Color Information Processing", *Journal of Mobile Multimedia*, 2024, Vol. 20, No. 3, pp. 679–698.

[30] V. Timchenko, V. Kreinovich, Y. Kondratenko, and V. Horbov, "Effectiveness Evaluations of Optical Color Fuzzy Computing", In: Y. P. Kondratenko (ed.), *Research Tendencies and Prospect Domains for AI Development and Implementation*, River Publishers, Denmark, 2024, pp. 129–151.

[31] V. Timchenko, V. Kreinovich, Y. Kondratenko, and V. Horbov, "Hybrid fuzzy-color computing based on optical logical architecture", In: C. Kahraman, S. Cevik Onar, S. Cebi, B. Oztaysi, A. C. Tolga, and I. Ucal Sari (eds.), *Intelligent and Fuzzy Systems, Proceedings of the 6th International Intelligent and Fuzzy Systems Conference INFUS 2024, Canakkale, Turkey, July 16–18, 2024*, Lecture Notes in Networks and Systems, Springer, Cham, Switzerland, Vol. 1090, pp. 266–274, https://doi.org/10.1007/978-3-031-67192-0_33

[32] V. L. Timchenko, Y. P. Kondratenko, V. Kreinovich, and O. Kosheleva, "Natural Color Interpretation of Interval-Valued Fuzzy Degrees", *Proceedings of the 20th World Congress of the International Fuzzy Systems Association IFSA'2023*, Daegu, South Korea, August 20–23, 2023, pp. 254–258.

[33] L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.