information

Article

Ordered Weighted Averaging (OWA), Decision Making under Uncertainty, and Deep Learning: How Is This All Related?

Vladik Kreinovich

Department of Computer Science, University of Texas at El Paso, El Paso, TX 79968, USA; vladik@utep.edu

Abstract: Among many research areas to which Ron Yager contributed are decision making under uncertainty (in particular, under interval and fuzzy uncertainty) and aggregation—where he proposed, analyzed, and utilized ordered weighted averaging (OWA). The OWA algorithm itself provides only a specific type of data aggregation. However, it turns out that if we allow several OWA stages, one after another, we obtain a scheme with a universal approximation property—moreover, a scheme which is perfectly equivalent to modern ReLU-based deep neural networks. In this sense, Ron Yager can be viewed as a (grand)father of ReLU-based deep learning. We also recall that the existing schemes for decision making under uncertainty are also naturally interpretable in OWA terms.

Keywords: ordered weighted averaging (OWA); decision making under uncertainty; deep learning

check for updates

Citation: Kreinovich, V. Ordered Weighted Averaging (OWA), Decision Making under Uncertainty, and Deep Learning: How Is This All Related? *Information* **2022**, *13*, 82. https://doi.org/10.3390/info13020082

Academic Editors: Luis Martínez López, Lesheng Jin, Zhen-Song Chen and Humberto Bustince

Received: 4 January 2022
Accepted: 8 February 2022
Published: 11 February 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

1. Introduction: How to Process Data

1.1. Need for Stages of Data Processing

In order to start analyzing the relation between OWA, deep learning, and decision making under uncertainty—several computer topics—let us first recall why computers are needed in the first place. The main objective of computers is to process data. Data processing is usually performed in several stages. Which processing algorithms should we select for each stage?

At each stage of a deterministic data processing, the result, y, is uniquely determined by the inputs, $x_1, \ldots, x_n$—this is what "deterministic" means. In mathematical terms, this means that, at each stage, we are computing a function of the inputs, $y = f(x_1, \ldots, x_n)$. Which functions should we select?

1.2. Linear Stages

The simplest possible functions are linear functions, as follows:

$$f(x_1, \ldots, x_n) = a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n.$$

They are ubiquitous in our description of the physical world, and there is a natural explanation for this ubiquity: most real-world dependencies are smooth, and in a reasonable neighborhood of each point, $x^{(0)} = \left(x_1^{(0)}, \ldots, x_n^{(0)}\right)$, a smooth function can be approximated well by a linear one, i.e., a function of the following form:

$$f(x_1, \ldots, x_n) \approx \left(x_1^{(0)}, \ldots, x_n^{(0)}\right) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \cdot \left(x_i - x_i^{(0)}\right);$$

See, for example, [1,2].

It is therefore reasonable to allow linear stages, where linear functions are computed.

1.3. Need for Nonlinear Stages

We cannot only use linear stages, since, in this case, all we would be able to compute are compositions of linear functions, and it is known that a composition of linear functions is always linear—while, in real life, many processes are nonlinear; see, e.g., [1,2].

Which nonlinear stages should we consider?

1.4. Importance of Symmetries and Invariance

To decide which nonlinear stages we should consider, let us recall how we acquire knowledge about the laws of the physical world in the first place.

For example, how did we learn that any object left in mid-air will fall down with the acceleration of 9.81 m/s^2? Someone (actually, Galileo) performed this experiment several times, and obtained this result. Then, this person moved to a different location, and obtained the same result, repeated this experiment at a later time—and got the same result. In other words, this person performed shifts in space, shifts in time, rotations, etc.—and the results did not change under these transformations. So, Galileo made a natural conclusion—that this phenomenon is invariant with respect to all such transformations, so if we repeat this experiment in another country, in another century, we will still observe the same behavior.

Such an invariance is behind all physical laws. When Ohm discovered their famous law—that voltage is proportional to current—he first found out that this relation holds in their lab. However, then, Ohm and others saw that the same relation happens in all locations, at all moments of time—so, it is indeed a universal physical law.

Physicists understand that invariances are the basis of all physical laws—see, e.g., [1,2]—to the extent that, in modern physics, new theories are usually proposed not in terms of differential equations—as in Newton's times—but in terms of the corresponding symmetries. Moreover, it turned out that many original physical theories, proposed first in terms of the differential equations, can actually be equivalently reformulated in terms of appropriate symmetries: this is true for Maxwell's equations that describe electromagnetism, Einstein's general relativity equations that describe gravity, and Schroedinger's equation that describes quantum phenomena, etc.—see, e.g., [3–5].

It is therefore reasonable to select nonlinear data processing stages which are invariant—and, ideally, invariant under as many natural transformations as possible.

1.5. Which Transformation Are Natural?

What are the natural transformations?

Some natural transformations come from the fact that we can use different measuring units to describe the same value of the physical quantity. For example, the height of 1.7 m is the same as the height of 170 cm. It is therefore reasonable to require that the result of data processing does not change if we change the value of the measuring unit—i.e., if we apply the corresponding transformation, $x \mapsto \lambda \cdot x$, for an appropriate value, λ. It is thus reasonable to add these transformations to our list of natural transformations.

The numerical values of many quantities, such as temperature or time, also depend on the choice of a starting point. If we take, as a starting point for measuring time, a moment x_0 years before the current year 0, then all the numerical values will change from their original values, x, to the new values, $x + x_0$. So, we should also add transformations, $x \mapsto x + x_0$, to our list of natural transformations.

In addition to such linear transformations, we often have nonlinear ones. For example, we can measure the strength of an earthquake by the amount of released energy, or by its logarithm—known as the Richter scale. We can measure the intensity of a sound by its energy, or in a logarithmic scale—in decibels. There are many other natural, nonlinear transformations, the only thing they all have in common is that they are all continuous and strictly increasing: what is larger in the original scale remains larger after the transformation. It is therefore reasonable to consider all continuous increasing transformations as natural.

Yet another class of natural transformations corresponds to the case when several inputs, x_i, correspond to the values of the same type of quantity—e.g., they are all moments of time, or they are all energy values. In this case, a reasonable idea is that we can permute them and obtain the exact same result. For example, if we bring together several objects of weights $x_1, \ldots, x_n$, the overall weight, $f(x_1, \ldots, x_n)$, should not depend on the order in which we brought these bodies together. Such permutations should therefore also be added to the list of natural transformations.

1.6. Which Functions Are Invariant with Respect to All Natural Transformations?

Now that we have decided which transformations to look into, let us analyze which computational stages—i.e., which functions $y = f(x_1, \ldots, x_n)$—are invariant with respect to all these transformations.

If all the values, x_i and y, represent different quantities—which can be independently transformed—then invariance would mean that for all continuous strictly increasing transformations, $t_i(x_i)$ and $t(y)$, if we have $y = f(x_1, \ldots, x_n)$, then we should have $t(y) = f(t_1(x_1), \ldots, t_n(x_n))$. Such invariance is not possible; indeed, if we take $t_i(x_i) = x_i$ and $t(y) = y + 1$, then this would mean that $y = f(x_1, \ldots, x_n)$ implies $y + 1 = f(x_1, \ldots, x_n)$, and that $y = y + 1$, and, thus, $0 = 1$—which is clearly impossible.

It is therefore reasonable to consider the case when all the quantities, x_i and y, represent the same quantity. In this case, invariance means that for any continuous strictly increasing function, $t(x)$, if we have $y = f(x_1, \ldots, x_n)$, then we should have $t(y) = f(t(x_1), \ldots, t(x_n))$. Since all the values, $x_1, \ldots, x_n$, represent the same quantity, invariance with respect to all natural transformations means also that the function $f(x_1, \ldots, x_n)$ should be invariant with respect to all permutations.

Now, we are ready to describe the class of all invariant functions.

Definition 1. *A continuous real-valued function, $f(x_1, \ldots, x_n)$ of n real variables is called* invariant *if it satisfies the following two conditions:*

- *for every continuous increasing function, $t(x)$, if we have $y = f(x_1, \ldots, x_n)$, then we have $t(y) = f(t(x_1), \ldots, t(x_n))$;*
- *for every permutation, $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, and for all possible values, x_i, we have*

$$f(x_{\pi(1)}, \ldots, x_{\pi(n)}) = f(x_1, \ldots, x_n).$$

There are known examples of invariant functions: namely, for each i from 1 to n, we can take $f(x_1, \ldots, x_n) = x_{(i)}$, where $x_{(i)}$ is the i-th of the values x_j, when we sort them in an increasing order, as follows:

- $x_{(1)}$ is the smallest of n values $x_1, \ldots, x_n$;
- $x_{(2)}$ is the second smallest of the values $x_1, \ldots, x_n$;
- $\ldots$;
- $x_{(n)}$ is the largest of the values $x_1, \ldots, x_n$.

Interestingly, there are the only invariant functions, as shown by the following result.

Proposition 1. *A function, $f(x_1, \ldots, x_n)$, is invariant if, and only if, it coincides with one of the functions $x_{(1)}, \ldots, x_{(n)}$.*

Proof. It is easy to check that all the functions $x_{(i)}$, are invariant, in the sense of the above definition. So, to complete our proof, it is sufficient to show that every invariant function is equal to one of the functions $x_{(i)}$.

Indeed, let $f(x_1, \ldots, x_n)$ be an invariant function. Let us first find what the value $f(1, \ldots, n)$ is. Our claim is that this value must be equal to one of the values $1, \ldots, n$. Indeed, if the value $v \stackrel{\text{def}}{=} f(1, \ldots, n)$ was different from these n numbers, then we could construct a continuous strictly increasing function, $t(x)$, for which $t(1) = 1, \ldots, t(n) = n$,

and $t(v) = v + \varepsilon$, for some small $\varepsilon > 0$. For example, we can form this function by using linear interpolation between the corresponding points. In this case, the first part of the definition of an invariant function would mean that $t(v) = v + \varepsilon = f(1, \ldots, n)$. However, since $v = f(1, \ldots, n)$, we obtain $v + \varepsilon = v$ and $\varepsilon = 0$—while we know that $\varepsilon > 0$.

Thus, the value $f(1, \ldots, n)$ is indeed equal to one of the values $1, \ldots, n$. Let us denote this value by i, then we obtain $f(1, \ldots, n) = i$.

Now let us consider the case when $x_1 < x_2 < \ldots < x_i < \ldots < x_n$. In this case, we can use linear interpolation to build a continuous strictly increasing piece-wise linear function $t(x)$ that maps 1 into x_1, 2 into x_2, ..., and n into x_n. For this transformation, $t(x)$, invariance and the fact that $f(1, \ldots, n) = i$ imply that $f(x_1, \ldots, x_n) = x_i$. In this case, $x_i = x_{(i)}$, so we have $f(x_1, \ldots, x_n) = x_{(i)}$.

If we have n values, $x_1, \ldots, x_n$, which are all different, then we can perform a permutation, π, that places these values in an increasing order, $x_{(1)} < \ldots < x_{(n)}$. For this permutation, the second part of the definition of invariance implies that $f(x_1, \ldots, x_n) = f\left(x_{(1)}, \ldots, x_{(n)}\right)$. For the sorted values, we already know that $f\left(x_{(1)}, \ldots, x_{(n)}\right) = x_{(i)}$, so we conclude that

$$f(x_1, \ldots, x_n) = x_{(i)}.$$

We have almost proved the proposition, the only thing that remains is to take care of the case when some of the values $x_1, \ldots, x_n$ are equal to each other. In this case, we can change the values a little bit—e.g., permute them so that they are in a non-decreasing order, and then take $X_i = x_i + i \cdot \delta$ for some small $\delta > 0$. For these changed values, we have $X_1 < X_2 < \ldots < X_n$, so $f(X_1, \ldots, X_n) = X_{(i)}$. Since the function $f(x_1, \ldots, x_n)$ is continuous, in the limit, $\delta \to 0$, we obtain the desired equality—$f(x_1, \ldots, x_n) = x_{(i)}$.

The proposition is proven. □

2. What Ordered Weighted Averaging Is and How It Is Related to Deep Learning

2.1. Reminder

In the previous section, we have shown that a reasonable way to form a computational process is to compose it of two types of stages, as follows:

- linear stages, when we compute a linear combination of the inputs

$$a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n;$$

- nonlinear stages, when we compute the i-th value $x_{(i)}$ in the increasing order.

If we allow only one stage, this is all we can compute: linear combinations and $x_{(i)}$. This is not enough—we can have more complex dependencies. So, we need more than one stage.

A natural question arises—what if we use two stages?

2.2. What If We Use Two Stages?

If we use two stages, then it does not make sense to have two linear stages one after another: this way, we compute the composition of two linear functions, and since this composition is also linear, this means that we can compute it in a single stage. Similarly, it does not make sense to have two nonlinear stages one after another: indeed, after the first nonlinear stage, we obtain sorted values, so an additional sorting of these results will not add any information.

So, we have two options, as follows:

- We can first perform a nonlinear stage and compute the values, $x_{(i)}$, and then perform a linear stage, i.e., compute a linear combination of these values,

$$a_0 + a_1 \cdot x_{(1)} + \ldots + a_n \cdot x_{(n)}.$$

In the statistical case, these are exactly L-estimators, frequently used robust statistical data processing techniques; see, e.g., [6]. In the general case, this formula describes exactly Yager's ordered weighted averaging (OWA); see, e.g., [7–9]. To be more precise, OWA is a particular case of this formula when $a_0 = 0$, $a_i \geq 0$, and

$$a_1 + \ldots + a_n = 1;$$

- Alternatively, we can first perform a linear stage and then perform a nonlinear stage. In this case, we obtain i-th element in the list of some linear combinations of inputs. As a particular case, we obtain the main units of deep learning (see, e.g., [10,11])—the rectified linear unit (ReLU) $\max(0, a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n)$, convolution (linear transformation), and max-pooling—when we compute $\max(x_1, \ldots, x_n)$.

If we consider only two stages, we can see the relation between OWA and deep learning, but the results are clearly different. In both cases, not all functions can be represented by these 2-stage computations.

To be able to compute any function with any given accuracy—i.e., to have a universal approximation property—we therefore need to have more than two stages. Here, things become interesting.

2.3. What If We Use Many Stages?

As we have mentioned, a linear stage (we will denote it by L) must be followed by a nonlinear stage (we will denote it by NL), and vice versa. Depending on which stage we start with, we have two possible sequences, as follows:

$$L \to NL \to L \to NL \ldots$$

or

$$NL \to L \to NL \to L \ldots$$

For both sequences, we can attach each NL stage to the previous L stage—as in deep neural networks—and obtain a sequence, as follows:

$$(L \to NL) \to (L \to NL) \ldots$$

or

$$NL \to (L \to NL) \to (L \to NL) \ldots$$

This way, we obtain a general ReLU-based neural network—which is known to have a universal approximation property; see, e.g., [11].

Alternatively, we can attach each L stage to the previous NL stage—as in OWA—and obtain a sequence, as follows:

$$L \to (NL \to L) \to (NL \to L) \ldots$$

or

$$(NL \to L) \to (NL \to L) \ldots$$

In other words, the same computations, as performed by a deep neural network, can be interpreted as several layers of OWA operations (and linear layers).

So, whatever we can compute with modern ReLU-based deep neural networks—and we can compute many important things [10,11]—can also be interpreted as a multi-stage application of OWA. In this sense, OWA can be viewed as a natural predecessor of ReLU-based deep neural networks—and Ron Yager can be viewed as a (grand)father of ReLU-based deep learning.

Important historical comment. The very idea of deep neural networks—i.e., neural networks that have more than one intermediate layers—can be traced to the mid-1960s [12,13]. However, at first, the successes of deep neural networks were limited—since the activation

functions originally used did not allow to researchers to uncover the full potential of having several layers. The modern spectacular successes of deep learning started when researchers realized the advantages of ReLU activation functions.

3. OWA and Decision Making under Uncertainty: Reminder

Another application area where OWA is important is decision making under uncertainty. This result is known, but, in view of the relation between OWA and ReLU-based machine learning, we feel that it is important to remind the readers about OWA's decision-making applications as well.

3.1. How to Make a Decision under Interval Uncertainty

In the ideal case, when we make a decision, we know the exact benefit, u_i, from each of the alternatives, i. In this case, a reasonable idea is to select the alternative with the largest benefit: $u_i \to \max$.

In practice, however, we usually do not know the exact values of the corresponding benefits. At best, for each alternative i, we know the bounds, $\underline{u}_i$ and $\overline{u}_i$, on the actual (unknown) value of the future benefit, u_i. In other words, all we know about the actual value, u_i, is that this value belongs to the interval, $[\underline{u}_i, \overline{u}_i]$. How can we make a decision in this situation?

Reasonable assumptions on rational decision making imply that we should select an alternative for which the following combination is the largest—$\alpha \cdot \overline{u}_i + (1-\alpha) \cdot \underline{u}_i$—where the value $\alpha \in [0,1]$ depends on the decision maker; see, e.g., [14–17]. This formula was first proposed by an economist Leo Hurwicz—who later received a Nobel Prize for this—and is thus known as the Hurwicz optimism–pessimism criterion. This name comes from the following facts:

- When $\alpha = 1$, then the Hurwicz combination is simply equal to $\overline{u}_i$. This means that when we make a decision, for each alternative, we only take into account the most optimistic value $\overline{u}_i$;
- When $\alpha = 0$, then the Hurwicz combination is simply equal to $\underline{u}_i$. This means that, when we make a decision, for each alternative, we only take into account the most pessimistic value, $\underline{u}_i$;
- When α is between 0 and 1, this means that take both optimistic and pessimistic values into account.

3.2. Hurwicz Criterion Is a Particular Case of OWA

It is known—this was noticed by Yager himself—that the above-described Hurwicz criterion is actually a particular case of OWA.

Indeed, how do we find the bounds, $\underline{u}_i$ and $\overline{u}_i$? We usually consider several possible scenarios, $s = 1, 2, \ldots, S$, and for each alternative decision, i, and for each scenario, s, we compute the resulting gain, u_{is}. Then, the following steps:

- we compute $\overline{u}_i$ as $\max(u_{i1}, u_{i2}, \ldots, u_{iS})$;
- we compute $\underline{u}_i$ as $\min(u_{i1}, u_{i2}, \ldots, u_{iS})$.

In these terms, Hurwicz's optimism–pessimism criterion means selecting the alternative with the largest possible value of the following quantity:

$$\alpha \cdot \max(u_{i1}, u_{i2}, \ldots, u_{iS}) + (1-\alpha) \cdot \min(u_{i1}, u_{i2}, \ldots, u_{iS}).$$

With respect to the inputs, $u_{i1}, u_{i2}, \ldots, u_{iS}$, this formula takes exactly the following OWA form:

$$\alpha \cdot u_{i(S)} + (1-\alpha) \cdot u_{i(1)}.$$

By the way, in this case, as in the original OWA, the weights α and $1-\alpha$ are both non-negative, and their sum is 1.

4. Conclusions

In the 1980s, Ron Yager proposed a new averaging operator—in which we first order the averaged inputs, and then take a linear combination of these inputs with weights, depending on the rank in this ordering. This OWA operator has turned out to be very useful in data processing and decision making.

In this paper, we show that OWA operators naturally appear if we consider algorithms consisting of stages of two types: linear combinations (the simplest possible operation) and nonlinear functions, which are invariant with respect to any linear or nonlinear re-scaling. If we only consider two such stages, we obtain, exactly, the OWA operators.

In general, when we allow more than two stages, the resulting algorithm is equivalent to applying several different OWA operators one after another. Such algorithms have a universal approximation property—they can approximate any continuous dependence on any bounded domain with any given accuracy. Interestingly, the class of functions that can be exactly computed in such computations is the same as the class of functions which can be exactly computed by the modern ReLU-based deep neural networks—in which linear combinations are intertwined with computing the ReLU function, $\max(0, x)$. Moreover, the number of consequent OWA operators is (almost) equal to the number of layers in the corresponding neural network. From this viewpoint, each deep neural network can be viewed as a sequential application of several OWA operators.

Funding: 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), and HRD-1834620 and HRD-2034030 (CAHSI Includes), and by the AT&T Fellowship in Information Technology. It was also supported by the program of the development of the Scientific Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478, and by a grant from the Hungarian National Research, Development and Innovation Office (NRDI).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: The author is thankful to Luis Martínez López, Lesheng Jin, Zhen-Song Chen, and Humberto Bustince, editors of this special issue, for their invitation, encouragement, and help, and to the anonymous referees for their useful suggestions.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Feynman, R.; Leighton, R.; Sands, M. *Feynman Lectures on Physics*; Addison Wesley: Boston, MA, USA, 2005.
2. Thorne, K.S.; Blandford, R.D. *Modern Classical Physics: Optics, Fluids, Plasmas, Elasticity, Relativity, and Statistical Physics*; Princeton University Press: Princeton, NJ, USA, 2017.
3. Finkelstein, A.; Kreinovich, V. Derivation of Einstein's, Brans-Dicke and other equations from group considerations. In *Sir Arthur Eddington Centenary Symposium*; Choque-Bruhat, Y., Karade, T.M., Eds.; World Scientific: Singapore, 1985; Volume 2, pp. 138–146.
4. Finkelstein, A.; Kreinovich, V.; Zapatrin, R.R. Fundamental physical equations uniquely determined by their symmetry groups. In *Global Analysis—Studies and Applications II*; Springer: Berlin/Heidelberg, Germany, 1986; pp. 159–170.
5. Kreinovich, V. Derivation of the Schroedinger equations from scale invariance. *Theor. Math. Phys.* **1976**, *3*, 282–285. [CrossRef]
6. Huber, P.J. *Robust Statistics*; Wiley-Interscience: New York, NY, USA, 2004.
7. Csiszar, O. Ordered Weighted Averaging Operators: A short review. *IEEE Syst. Man, Cybern. Mag.* **2021**, *7*, 7–12. [CrossRef]
8. Yager, R.R.; Kacprzyk, J. *The Ordered Weighted Averaging Operators: Theory and Applications*; Kluwer Academic Publishers: Dordrecht, The Netherlands, 1997.
9. Yager, R.R.; Kacprzyk, J.; Belyakov, G. *Recent Developments in the Ordered Weighted Averaging Operators: Theory and Practice*; Springer: Cham, Switzerland, 2011.
10. Ekman, M. *Learning Deep Learning: Theory and Practice of Neural Networks, Computer Vision, Natural Language Processing, and Transformers Using TensorFlow*; Addison-Wesley: Boston, MA, USA, 2021.
11. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Leaning*; MIT Press: Cambridge, MA, USA, 2016.
12. Ivakhnenko, A.G.; Lapa, V.G. *Cybernetic Predicting Devices*; Naukova Dumka: Kiev, Ukraine, 1965. (In Russian)
13. Ivakhnenko, A.G.; Lapa, V.G. *Cybernetics and Forecasting Techniques*; Elsevier: New York, NY, USA, 1967.
14. Hurwicz, L. *Optimality Criteria for Decision Making Under Ignorance*; Cowles Commission Discussion Paper, Statistics, No. 370; Cowles Commission for Research in Econoimics: Chicago, IL, USA, 1951.

15. Kreinovich, V. Decision making under interval uncertainty (and beyond). In *Human-Centric Decision-Making Models for Social Sciences*; Guo, P., Pedrycz, W., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; pp. 163–193.
16. Luce, R.D.; Raiffa, R. *Games and Decisions: Introduction and Critical Survey*; Courier Corporation: Dover, NY, USA, 1989.
17. Tockel, W. (Ed.) *Social Design: Essays in Memory of Leonid Hurwicz*; Springer: Cham, Switzerland, 2019.