Uncertainty: Ideas Behind Neural Networks Lead Us Beyond KL-Decomposition and Interval Fields*

1st Michael Beer
Institute for Risk and Reliability
Leibniz University Hannover
Hannover, Germany
beer@irz.uni-hannover.de

2nd Olga Kosheleva
Department of Teacher Education
University of Texas at El Paso
El Paso, Texas, USA
olgak@utep.edu

3rd Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
El Paso, Texas, USA
vladik@utep.edu

***Abstract*—In many practical situations, we know that there is a functional dependence between a quantity q and quantities $a_1, \ldots, a_n$, but the exact form of this dependence is only known with uncertainty. In some cases, we only know the class of possible functions describing this dependence. In other cases, we also know the probabilities of different functions from this class – i.e., we know the corresponding random field or random process. To solve problems related to such a dependence, it is desirable to be able to simulate the corresponding functions, i.e., to have algorithms that transform simple intervals or simple random variables into functions from the desired class. Many of the real-life dependencies are very complex, requiring a large amount of computation time even if we ignore the uncertainty. So, to make simulation of uncertainty practically feasible, we need to make sure that the corresponding simulation algorithm is as fast as possible. In this paper, we show that for this objective, ideas behind neural networks lead to the known Karhunen-Loéve decomposition and interval field techniques – and also that these ideas help us go – when necessary – beyond these techniques.**

***Index Terms*—neural networks, interval fields, Karhunen-Loéve decomposition**

I. General Problem: Need for Simulations under Uncertainty

A. Dependencies are ubiquitous

In a nutshell, the main purpose of engineering is to make the world better: whether:

- by controlling natural phenomena (e.g., in damming flooding rivers) or
- by coming up with devices (and ways to use them) that will make the world better for us.

For this purpose, we need to describe the current state of world, and to predict how the state of the world will change under our actions.

Describing the state of world means describing, at each spatial point (x, y) (or, in the 3-D case, (x, y, z)), the values of all relevant quantities q at this point. In other words, we want to know the functions $q(x, y)$ or $q(x, y, z)$.

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).

To decide how to change the state of the world, we need to know how the resulting state – i.e., how the resulting values of different quantities – will depend on the parameters $a_1, \ldots, a_n$ characterizing possible actions. In other words, we want to know the dependence $q(a_1, \ldots, a_n)$.

In all these cases, we want to known a function – or functions.

B. Need to take uncertainty into account

Most of the information about the world comes from measurements, and measurements are never absolutely accurate, measurements results are always approximate; see, e.g., [16]. Similarly, the information about dependencies comes from measurements. As a result:

- we never know the exact function $q(a_1, \ldots, a_n)$;
- instead, there is a whole class $\mathcal{F}$ of different functions which are all consistent with the known knowledge.

In many cases:

- in addition to knowing this set $\mathcal{F}$,
- we also know the frequencies (= probabilities) with which different functions from the class $\mathcal{F}$ are expected to occur in similar situations.

In mathematical terms, this situation is known as a *random field*, or, in the 1-D case, a *random process*; see, e.g., [22].

C. Need to simulate

One of the most useful methods of studying the world – especially when the equations are too complicated to solve explicitly – is by computer simulation of the corresponding processes.

D. Need to take uncertainty into account during simulations

To make situations realistic, we need to take uncertainty into account. For this purpose, it is desirable to come up with simulations that would:

- produce different functions from the corresponding class $\mathcal{F}$,
- and, if we know the frequencies (probabilities) of different function, produce different functions from the class $\mathcal{F}$ with exactly this frequency.

E. What can be the input to such simulations

To simulate different functions from the given class, a reasonable idea is to use simplest 1-D type of the corresponding uncertainty as inputs:

- In situations when we do not know the probabilities, when we only know the class (= set) F, a natural idea is to use numbers for which we also know only they belong to some set of real numbers. The simplest such set is an interval $[-1, 1]$. So, a natural idea is to use, as inputs, numbers $c_1, \ldots, c_N$ from the interval $[-1, 1]$.
- In situations when we know the probabilities, a reasonable idea is to use independence 1-D random variables c_i with some simple distributions: e.g., uniformly distributed on the interval $[0, 1]$ or normally distributed with mean 0 and standard deviation 1.

So, we need algorithms that would transform the values

$$c = (c_1, \ldots, c_N)$$

into the corresponding functions $f_c(a_1, \ldots, a_m)$ from the class $\mathcal{F}$.

F. Problem: we need feasible-to-compute algorithms

The dependencies from the class $\mathcal{F}$ are often rather complex, their computation is very time-consuming even if we ignore the uncertainty. For example, accurately predicting tomorrow's weather require spending several hours on a high-performance computer to solve the corresponding system of partial differential equations.

Taking uncertainty into account would make the computational problems even more complex and thus, requiring even more computation time. It is therefore desirable to come up with the fastest possible simulation algorithms.

G. What we do in this paper

In this paper, we show that the main ideas behind neural networks help explain why KL-decomposition and interval fields are efficient in such uncertainty-related simulations.

We also show that these ideas help to go beyond these techniques – and we also explain why we need to go beyond these two techniques.

II. Main Ideas Behind Neural Networks: A Brief Reminder

A. What we do in this section

In this section, we recall the main ideas behind neural networks – not all the ideas, but only those that will be helpful in solving our problem.

Namely, what we will recall is how the need to make computations fast naturally lead to neural networks – specifically, to traditional neural networks [3], [11], [12], that were prevalent before the deep learning revolution (see, e.g., [7], [11], [13]).

B. What elementary steps can we use to speed up computations

Every computation is composed of elementary steps. To speed up computations, a reasonable idea is to perform several computational steps in parallel, on different processors. This is how high-performance computers work – in particular, high-performance computers that predict weather and solve other complex computational problems.

For such computations, the overall computation time is the sum of computation times of different computational steps (performed in parallel). Thus, to speed up computations, we need:

- to make these steps as fast as possible, and
- to minimize the overall number of such steps.

C. What are the fastest possible steps?

On each computational step, we transform some inputs $x_1, \ldots, x_n$ into one or more outputs y. The output y is usually uniquely determined by the inputs, so, in mathematical terms, we compute the value $f(x_1, \ldots, x_n)$ of an appropriate function.

Which functions are the easiest to compute? In general, functions can be linear and non-linear. Clearly, linear functions

$$y = w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n$$

are the easiest (and thus, the fastest) to compute. So linear functions must be among the corresponding elementary computational steps.

D. Need for nonlinear steps

We cannot have only linear steps. If we only use linear steps, then what we will compute is a composition of linear functions – and this composition is always linear. On the other hand, many real-life processes are non-linear; see, e.g., [6], [21]. Thus, in addition to linear computational steps, we also need to use some non-linear elementary computational steps.

E. Which nonlinear steps shall we use?

Which nonlinear functions should we use? In general:

- the more inputs we use,
- the more time is needed for computations.

Thus, the fastest to compute are the nonlinear functions that have the smallest possible number of inputs – only one. So:

- in additional linear elementary computational steps,
- we should also use step that consist of applying a nonlinear function $y = f(x)$ to a single input.

F. Layers

Computations on a parallel computer comes in what is called layers:

- first, all the processors perform one type of operations,
- then they perform other types of operations, etc.

As we have argued, each layer must consist of:

- either computing linear functions,
- or computing non-linear functions of one variable.

Let us denote:

- a linear layer by L, and
- a nonlinear layer by N.

G. Layers must be interleaving

Our goal is to speed up computations. So, it does not make sense to have a linear layer following a linear layer. Indeed, in such a $L-L$ configuration, all we are computing are compositions of linear functions – which, as we have mentioned, are also linear. Thus, we could as well compute these functions faster, by using a single linear layer instead of two.

Similarly, it does not make sense to have a nonlinear layer following a nonlinear layer. Indeed, in such a $N-N$ configuration, all we are computing are compositions of functions of one variable, i.e., expressions $y = f(g(x))$ which are also simply functions of one variable. Thus, we could as well compute these functions faster, by using a single nonlinear layer instead of two.

So, layers must interleave:

- a linear layer (which is not final) must be followed by a nonlinear layer, and
- a nonlinear layer (which is not final) must be followed by a linear layer.

H. How many layers do we need?

As we have mentioned, to speed up computations, we need to use the smallest possible number of layers. Let us analyze how many layers we need.

I. One layer is not sufficient

With one layer, we can compute either linear functions, or nonlinear functions of a single input. In practice, however, many real-life dependencies are nonlinear, and have more than one input.

Thus, one layer is not enough.

J. What about two layers: two options

In the case of two layers, we can have:

- either a linear layer followed by a nonlinear layer,
- or a nonlinear layer followed by a linear layer.

Let us show that in both cases, we cannot cover all possible dependencies. Specifically, we will show that we are not even able to ocver the product

$$y = x_1 \cdot x_2.$$

K. $NL-L$ configuration

In the case of $NL-L$ configuration, for $n=2$ inputs:

- In the first layer, we compute the values
$$f_i(x_1) \text{ and/or } g_j(x_2).$$
- In the second layer, we compute a linear combination of these results, i.e., the value
$$w_0 + w_1 \cdot f_1(x_1) + w_2 \cdot f_2(x_1) + \ldots + v_1 \cdot g_1(x_2) + v_2 \cdot g_2(x_2) + \ldots$$

We can combine:

- all the value depending on x_1 into a single expression, and
- all the values depending on x_2 into a single expression.

Thus, we get

$$y = F_1(x_1) + F_2(x_2),$$

where we denoted

$$F_1(x_1) \stackrel{\text{def}}{=} w_0 + w_1 \cdot f_1(x_1) + w_2 \cdot f_2(x_1) + \ldots$$

and

$$F_2(x_2) = v_1 \cdot g_1(x_2) + v_2 \cdot g_2(x_2) + \ldots$$

Let us show that the expression $x_1 \cdot x_2$ cannot be described in the form $F_1(x_1) + F_2(x_2)$. Indeed, if we could have

$$x_1 \cdot x_2 = F_1(x_1) + F_2(x_2),$$

then:

- from the fact that $0 \cdot x_2 = 0$ for all x_2, we conclude that
$$0 = F_1(0) + F_2(x_2),$$
i.e., that
$$F_2(x_2) = -F_1(0) = \text{const};$$
- similarly, from the fact that $x_1 \cdot 0 = 0$ for all x_1, we conclude that
$$0 = F_1(x_1) + F_2(0),$$
i.e., that
$$F_1(x_1) = -F_2(0) = \text{const}.$$

Thus, the sum $F_1(x_1) + F_2(x_2)$ of two constant functions is also a constant, not depending on x_i – and cannot thus be equal to the product $x_1 \cdot x_2$.

L. $L-NL$ configuration

In the case of $L-NL$ configuration:

- in the first layer, we compute a linear combination
$$w_0 + w_1 \cdot x_1 + w_2 \cdot x_2,$$
and
- in the second later, we apply a nonlinear function $f(x)$ of one variable to this linear combination, resulting in
$$F(x_1, x_2) = f(w_0 + w_1 \cdot x_1 + w_2 \cdot x_2).$$

Let us show that the product $x_1 \cdot x_2$ cannot be equal to such an expression.

Indeed, in the case of such an equality

$$x_1 \cdot x_1 = F(x_1, x_2) = f(w_0 + w_1 \cdot x_1 + w_2 \cdot x_2),$$

we must have $w_1 \neq 0$ and $w_2 \neq 0$ – otherwise the expression $F(x_1, x_2)$ will not depend on x_1 or on x_2. Now, for each x_2, we have

$$0 = 0 \cdot x_2 = F(w_0 + w_2 \cdot x_2).$$

Since $w_2 \neq 0$, the expression

$$w_0 + w_2 \cdot x_2$$

can take any real value x, so we have $F(x) = 0$ for all x – and therefore, it is not possible to have

$$0 = F(1 \cdot 1) = 1 \cdot 1.$$

So, 2 layers are not enough, we must have at least 3 layers.

M. Which 3-layer configuration is the fastest

Since layers must interleave, we can have two possible 3-layer configurations: $L - NL - L$ and $NL - L - NL$.

- In the first case, we have one nonlinear layer and two linear layers.
- In the second case, we have two nonlinear layers and one linear layer.

Since, as we have mentioned, a linear layer is faster than a nonlinear one, the $L - NL - L$ configuration is faster.

N. A general formula for the $L - NL - L$ configuration

In the first later, each processor k computes a linear combination of inputs:

$$z_k = w_{k0} + w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n.$$

In the second layer, we apply some nonlinear function of each outputs y_k of the first layer, computing $y_k = f_k(y_k)$ for some functions $f_k(x)$. Finally, on the last linear layer we compute a linear combination of the values y_k:

$$y = W_0 + W_1 \cdot y_1 + \ldots + W_K \cdot y_K,$$

i.e., the value

$$y = W_0 + W_1 \cdot f_1(w_{10} + w_{11} \cdot x_1 + \ldots + w_{1n} \cdot x_n) + \ldots + W_K \cdot f_K(w_{K0} + w_{K1} \cdot x_1 + \ldots + w_{Kn} \cdot x_n). \quad (1)$$

Comment. It should be mentioned that this is exactly what the traditional neural networks compute. The only difference is that they usually use the same function $f_k(x) = f(x)$ for all intermediate results.

O. Three layers are sufficient

The fact that a 3-layer configuration is sufficient to represent any continuous function on bounded domain with any desired accuracy follows from the fact that each such function can be represented as a Fourier transform, i.e., as a linear combination of sines of linear combinations of the inputs.

This potentially infinite linear combination can be approximated, with any given accuracy, by a finite sum of sines, i.e., by the expression (1) for

$$f_1(x) = \ldots = f_K(x) = \sin(x).$$

III. Main Ideas Behind Neural Networks Explain KL-Decomposition and Interval Fields

A. What we want: a reminder

We want to have a fast algorithm that would transform values $c = (c_1, \ldots, c_N)$ that represent simple uncertainties – interval or random – into the corresponding values

$$f_c(a_1, \ldots, a_n).$$

B. Which are the fastest possible algorithms for this purpose

As we have mentioned earlier, the fastest possible algorithms are linear. Thus, we arrive at the following expression which is linear in c_i:

$$f_c(a_1, \ldots, a_m) = w_0(a_1, \ldots, a_n) + c_1 \cdot w_1(a_1, \ldots, a_n) + \ldots + c_N \cdot w_N(a_1, \ldots, a_n). \quad (2)$$

C. Interval case: main observation

In situations when each c_i is taking values from the interval $[-1, 1]$, the expression (2) becomes a particular case of so-called *interval fields*, a useful techniques for analyzing such uncertainty; see, e.g., [1], [2], [9], [23].

D. Interval case: detailed analysis

In general, an interval field is defined as the class of all functions of type (2) when each c_i are in a given interval $[\underline{c}_i, \overline{c}_i]$. Let us show that this general definition can be reduced to the above case, when all the values come from the interval $[-1, 1]$.

Indeed, each interval $[\underline{c}_i, \overline{c}_i]$ can be represented as

$$[\widetilde{c}_i - \Delta_i, \widetilde{x}_i + \Delta_i],$$

where

$$\widetilde{c}_i \stackrel{\text{def}}{=} \frac{\underline{c}_i + \overline{c}_i}{2}$$

is the interval's midpoint, and

$$\Delta_i \stackrel{\text{def}}{=} \frac{\overline{c}_i - \underline{c}_i}{2}$$

is the interval's half-width, also known – taking into account that an interval is a natural 1-D analogue of a 2-D disk – as its radius.

In these terms, each number c_i from the interval $[\underline{c}_i, \overline{c}_i]$ can be represented as $c_i = \widetilde{c}_i + r_i \cdot \Delta_i$, where r_i takes all possible values from the interval $[-1, 1]$. Thus, the expression (2) takes the form

$$f_c(a_1, \ldots, a_m) = w_0(a_1, \ldots, a_n) + (\widetilde{c}_1 + r_1 \cdot \Delta_1) \cdot w_1(a_1, \ldots, a_n) + \ldots + (\widetilde{c}_N + r_N \cdot \Delta_N) \cdot w_N(a_1, \ldots, a_n),$$

i.e., the form

$$f_r(a_1, \ldots, a_n) = W_0(a_1, \ldots, a_n) + r_1 \cdot W_1(a_1, \ldots, a_m) + \ldots + r_N \cdot W_N(a_1, \ldots, a_m),$$

where we denoted

$$W_0(a_1, \ldots, a_n) \stackrel{\text{def}}{=} w_0(a_1, \ldots, a_n) + \widetilde{c}_1 \cdot w_1(a_1, \ldots, a_n) + \ldots + \widetilde{c}_N \cdot w_N(a_1, \ldots, a_n)$$

and

$$W_k(a_1, \ldots, a_n) \stackrel{\text{def}}{=} \Delta_k \cdot w_k(a_1, \ldots, a_n),$$

and each variable r_k takes all possible values from the interval

$$[-1, 1].$$

E. Computational comment

To make computations faster, it is desirable to have a representation (2) that uses as few terms N as possible. Such a reduction is possible. Indeed, in the linear space of all functions, terms $c_k \cdot w_k(a_1, \ldots, a_n)$ corresponding to different values $c_i \in [-1, 1]$ form a straight line formed by endpoints:

- $w_k(a_1, \ldots, a_n)$ (corresponding to $c_k = 1$) and
- $-w_k(a_1, \ldots, a_n)$ (corresponding to $c_k = -1$).

The set of all the functions of type (2) – the sum of all possible sums of such terms – is known as the *Minkowski sum* of such intervals. A Minkowski sum of straight line intervals is known as a *zonotope*, and it is known that zonotopes can be approximated by zonotopes with smaller number of terms; see, e.g., [4], [10], [20].

F. Probabilistic case

In the case when c_i are independent random variables, a particular case of the expression (2) is Karhunen-Loéve (KL) representation of a random field – to be more precise, as a finite approximation to a generic potentially infinite KL representation; see, e.g., [19], [22].

G. Conclusion of this section

Thus, the main ideas behind neural networks indeed explain the empirical success of KL-decomposition and interval field techniques.

IV. Need to Go Beyond KL-Representations and Interval Fields

A. Which random processes can be represented by KL representations

In the general probabilistic case, the expression (2) is a sum of a large number of independent random variables

$$c_i \cdot w_i(a_1, \ldots, a_n).$$

In general, each of these terms is reasonably small – otherwise, if a few of these terms were domineering, we could have left only these terms and ignore the others.

It is known that, under reasonable conditions, the distribution of the sum of a large number of small independent random variable is close to Gaussian. This follows from the Central Limit Theorem – see, e.g., [18] – according to which the distribution of such a sum tends to Gaussian when the number of terms increases.

Thus, the expression (2) only works for Gaussian random fields – this is how KL representation is usually used.

B. Need to go beyond KL representations

As we have mentioned, the uncertainty comes from measurement errors. It is known that:

- while many measurement errors are indeed normally distributed,
- almost a half of them is not; see, e.g., [**?**], [15].

To describe such non-Gaussian distributions, we need to go beyond KL-representations.

C. Non-probabilistic case; which classes of functions can be represented by interval fields

Each segment $[-1, 1] \cdot w_i(a_1, \ldots, a_n)$ is a convex set. It is known the Minkowski sum of convex sets is a convex set; see, e.g., [17].

Thus, only convex sets of functions can be represented in the form (2).

D. Need to go beyond interval fields

Not all sets of possible values are convex. Indeed:

- it is known that a nonlinear transformation, in general, transforms convex sets into non-convex ones, and,
- as we have mentioned, many real-life transformations are nonlinear.

V. Ideas Behind Neural Networks Explain How to Go Beyond KL-Decomposition and Interval Fields

A. What is a natural next step after linear transformations: reminder

As we have mentioned earlier, the need to perform computations fast leads to a layered computation scheme, with a small number of linear (L) and nonlinear (N) layers, in which each element of a nonlinear layer applies a nonlinear function of one variable to its input.

In the above text, we showed that KL-decomposition and interval fields correspond to using just one linear layer. Since, as we have shown, in many practical situations, we need to go beyond these techniques, we therefore need to turn to the next fastest approach, when we have two layers.

Since the layers must interleave, we have two possible 2-layer configurations:

- $NL - L$ and
- $L - NL$.

Let us analyze these two configurations one by one.

B. $NL - L$ approach: analysis

For our problem, $NL - L$ approach means that:

- we first apply some nonlinear transformations $c_i \mapsto f_i(c_i)$ to the inputs c_i, and
- then perform a linear transformation:

$$f_c(a_1, \ldots, a_m) = w_0(a_1, \ldots, a_n) + f_1(c_1) \cdot w_1(a_1, \ldots, a_n) + \ldots + f_N(c_N) \cdot w_N(a_1, \ldots, a_n). \quad (3)$$

Will this help? Not really:

- In the probabilistic case, the variables $f_i(c_i)$ are still independent, so the same Central Limit Theorem still shows that, as a result, we get a Gaussian field or a Gaussian process.
- In the interval case, for continuous functions $f_i(c_i)$, the range of this function when $c_i \in [-1, 1]$ is still an interval, so we still get an interval field.

C. $L - NL$ approach: formulas

Since we cannot use $NL - L$ approach, a natural idea is to use the $L - NL$ approach, i.e., to apply a nonlinear function $f(x)$ to the result (2) of linear processing. This way, we get the following result:

$$f_c(a_1, \ldots, a_m) = f\left(w_0(a_1, \ldots, a_n) + \sum_{i=1}^{N} c_i \cdot w_i(a_1, \ldots, a_n)\right). \quad (4)$$

D. This approach works

Empirical results show that the formula (4) – corresponding to the $L - NL$ approach – indeed leads to a very good description of non-Gaussian probabilistic uncertainty; see, e.g., [5], [8].

We hope that it will be as useful in describe non-convex classes of functions.

E. What next?

While so far, the $L - NL$ approach works well, as we have mentioned, the $L - NL$ approach cannot describe all possible dependencies. So, at some point, we will encounter practical situations when this approach needs to be replaced by a more accurate one. How can we do it?

Again, the above-described main ideas behind neural networks provide a natural answer: we need to consider 3-layer $L - NL - L$ approach, i.e., to consider expressions of the type

$$f_c(a_1, \ldots, a_m) = W_0 + W_1 \cdot f_1\left(w_{10}(a_1, \ldots, a_n) + \sum_{i=1}^{N} c_i \cdot w_{1i}(a_1, \ldots, a_n)\right) + \ldots + W_K \cdot f_K\left(w_{K0}(a_1, \ldots, a_n) + \sum_{i=1}^{N} c_i \cdot w_{Ki}(a_1, \ldots, a_n)\right). \quad (5)$$

This approach already have the universal approximation property – so it can describe the corresponding class of functions (and, if appropriate, the probability distribution on this class of functions) with any desired accuracy.

References

[1] D. Betancourt and R. L. Muhanna, "Deep interval neural network in computational mechanics", Proceedings of the 9th International Workshop on Reliable Engineering Computing REC'2021, Taormina, Italy, May 16–20, 2021, pp. 11–26.

[2] D. Betancourt, R. Muhanna, and R. Mullen, "Interval field for spatially and temporally dependent uncertainty – machine learning approach", In: Proceedings of the 8th International Workshop on Reliable Engineering, REC'2018, Liverpool, UK, July 16–18, 2018.

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

[4] J. Bourgain, J. Lindenstrauss, and V. Milman, "Approximation of zonoids by zonotopes", Acta Mathematica, vol. 162, no. 1–2, pp. 73–141, 1989.

[5] M. G. R. Faes, M. Broggi, K.-K. Phoon, and M. Beer, "Distribution-free P-box processes: definition and simulation", Proceedings of the 9th International Workshop on Reliable Engineering Computing REC'2021, Taormina, Italy, May 16–20, 2021, pp. 380–394.

[6] R. Feynman, R. Leighton, and M. Sands, The Feynman Lectures on Physics, Boston, Massachusetts: Addison Wesley, 2005.

[7] I. Goodfellow, Y. Bengio, and A. Courville, Deep Leaning, Cambridge, Massachusetts: MIT Press, 2016.

[8] M. Grigoriu, "Simulation of stationary non-Gaussian translation processes", *Journal of Engineering Mechanics*, vol. 124, no. 2, pp. 121–126, 1998.

[9] M. Imholz, M. Faes, J. Cerneels, D. Vandepitte, and D. Moens, "On the comparison of two novel interval field formulations for the representation of spatial uncertainty" In: Proceedings of the 7th International Workshop on Reliable Engineering Computing REC'2016, Bochum, Germany, June 15–17, 2016, pp. 367–378.

[10] O. Kosheleva and V. Kreinovich, "Low-complexity zonotopes can enhance uncertainty quantification (UQ)", Proceedings of the 4th International Conference on Uncertainty Quantification in Computational Sciences and Engineering UNCECOMP'2021, Athens, Greece, June 28–30, 2021.

[11] V. Kreinovich, "From traditional neural networks to deep learning: towards mathematical foundations of empirical successes", In: S. N. Shahbazova, J. Kacprzyk, V. E. Balas, and V. Kreinovich (eds.), Recent Developments and the New Direction in Soft-Computing Foundations and Applications: Selected Papers from the World Conference on Soft Computing, Baku, Azerbaijan, May 29–31, 2018, Cham, Switzerland: Springer, 2021, pp. 387–397.

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

[13] V. Kreinovich and O. Kosheleva, "Optimization under uncertainty explains empirical success of deep learning heuristics", In: P. Pardalos, V. Rasskazova, and M. N. Vrahatis (eds.), Black Box Optimization, Machine Learning and No-Free Lunch Theorems, Cham, Switzerland: Springer, 2021, pp. 195–220.

[14] P. V. Novitskii and I. A. Zograph, Estimating the Measurement Errors, Leningrad: Energoatomizdat, 1991, in Russian.

[15] A. I. Orlov, How often are the observations normal?, *Industrial Laboratory*, vol. 57, no. 7, pp. 770–772, 1991.

[16] S. G. Rabinovich, Measurement Errors and Uncertainties: Theory and Practice, New York: Springer, 2005.

[17] R. T. Rockafeller, Convex Analysis, Princeton, New Jersey: Princeton University Press, 1997.

[18] D. J. Sheskin, Handbook of Parametric and Non-Parametric Statistical Procedures, London, UK: Chapman & Hall/CRC, 2011.

[19] P. Spanos and R. Ghanem, "Stochastic finite element expansion for random media", *Journal of Engineering Mechanics*, vol. 115, no. 5, pp. 1035–1053, 1989.

[20] T. Tao, Exploring the toolkit of Jean Bourgain, Bulletin on the American Mathematical Society, vol. 58, pp. 155–171, 2021.

[21] K. S. Thorne and R. D. Blandford, Modern Classical Physics: Optics, Fluids, Plasmas, Elasticity, Relativity, and Statistical Physics, Princeton, New Jersey: Princeton University Press, 2017.

[22] E. Vanmarcke, Random Fields: Analysis and Synthesis, Cambridge, Massachusetts: MIT Press, 1983.

[23] W. Verhaeghe, W. Desmet, D. Vandepitte, and D. Moens, "Uncertainty assessment in random field representations: an interval approach", Proceedings of the 30th Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'2011, El Paso, Texas, March 18–20, 2011.