Which AI/ML Techniques to Select for Applications to Decision Making: Towards Theoretical Explanations of Empirical Discoveries

Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
vladik@utep.edu

Abstract

This chapter provides theoretical explanations for selecting best machine learning technique for decision making. Mostly, it concentrates on the theoretical explanation of why empirically, ReLU activation function works the best. There are many possible criteria for selecting such a function: we may look for the fastest-to-compute function, we may look for the function that is the most interpretable, etc. Somewhat unexpectedly, all these criteria lead to the exact same conclusion – that under several reasonable optimality criteria, ReLU is optimal – e.g., the most computationally efficient. After describing these results, we explain why all these criteria lead to the same selection. We finish this chapter with an overview of how similar techniques can help with explaining other empirically successful features of deep learning – and of data processing in general.

1 Machine learning techniques: empirical successes and related theoretical challenges

What is deep learning – an empirically successful machine learning technique. Recent decades have shown spectacular applications of machine learning techniques, especially techniques of deep neural networks; see, e.g., [30].

Deep neural networks are particularly successful. In this technique, data processing is performed by *neurons*, devices that take several inputs $x_1, \ldots, x_n$ and compute the value $s(w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n)$, where w_i are numbers called *weights* and $s(z)$ is a function known as *activation function*. In deep learning, the most widely used activation function is $s(z) = \max(0, z)$, which is known as the *Rectified Linear Unit* (*ReLU*, for short).

Some neurons take, as inputs, the inputs to the data processing problem. Other neurons use, as inputs, the output signals of other neurons, etc. The output signal of one of the neurons is then returned as the result of data processing.

In the past, researchers considered what is called *3-layer* neural networks, where:

- first, all the input signals are processed by several neurons in the so-called hidden layer, and
- then the output – linear – neuron transforms the outputs of hidden-layer neurons into the result of data processing.

However, lately, it turned out that neural networks with more layers – called *deep* – provide much better results than traditional "shallow" neural networks.

Each empirical success is a theoretical challenge. Some of these successes of deep learning come from theoretical analysis, but most of them are largely empirical.

From the theoretical viewpoint, every time we have a successful heuristic for which no theoretical explanation is known, we have a challenge: to explain this empirical success.

But why would anyone care about theory if we already have empirical successes? One may wonder: why should anyone care about theoretical explanations if we already have empirical successes? Theory is good when it leads to new results – but if results are already there, why care about explaining this success? If aspirin successfully cures a headache, why would anyone be interested in how exactly it works – which was, by the way, a mystery for many decades.

There are two answers to this question. First, when empirically, e.g., some activation function works better than others, this does not necessarily mean that this is indeed the best possible function: empirically, we can test only finitely many functions, and there are infinitely many of them. So, without a theoretical analysis, we cannot be sure that one of the functions that we did not try is actually better than the one we are using now. And maybe a theoretical analysis will find out that the empirically best function is not actually optimal – and this analysis will help us find a better function.

Second, theoretical results often apply not only to a specific application, but also to many other applications – including those that we have not yet empirically tried. This can provide us a guidance to solving other problems.

What we do in this chapter. In this chapter, we provide possible theoretical explanations for several empirically successful features of deep learning.

To come up with these explanations, we will recall what people want from machine learning – especially when it is applied to decision making. For each of the resulting features, we will formulate the corresponding objective, and then we show how to solve the corresponding optimization problem.

We will mostly concentrate on the problem of selecting the best activation function – i.e., mostly, on explaining why empirically, the ReLU activation function $s(x) = \max(0, z)$ is the most effective.

Our results are somewhat unexpected – but we explain this unexpected feature as well. Our analysis has led to some unexpected results. Let us explain in what sense our results are unexpected.

Intuitively, when there are several objectives, one expects a trade-off. For example, for optimization algorithms, there is usually a trade-off between the quality of the result and the computation time: if we want better results, we need to spend more computation time and, vice versa, if we want to decrease computation time, the quality of the results decreases.

In contrast, in our case, all reasonable criteria lead to the exact same conclusion – that under each of these criteria, ReLU is optimal. This sounds strange – and explaining this was yet another challenge for us – but we have found an explanation for this phenomenon. This explanation will be provided as well.

Finally, at the end of the paper, we will briefly explain how these results can be (and often were) extended to explaining other empirically successful features of deep learning.

Comment. In this chapter, we cover many different explanations. Many of them are based on complex (and lengthy) mathematical proofs. We realize that it is not realistically possible to present all these proofs in this chapter – if we did that, we would get a whole book, not a chapter. So, in this chapter, we will only provide a few shorter proofs. For most other results, we will describe the main ideas behind the corresponding proofs – and provide links to papers where these proofs are given in detail.

2 What do we expect from a machine learning application to decision making?

Let us first recall what we want from such an application. In general, in a decision making problem, we start with a real-world situation, and we want to process this information to come up with a decision that will be, in some reasonable sense, optimal – or at least adequate – in this situation. Usually, real-life decision making comes in two steps:

- first, we try to understand the situation as accurately as possible, and
- then, based on this understanding, we make a decision.

For example, to make a medical decision, a doctor usually:

- first orders some tests to determine what exactly is the problem, and
- then, based on the resulting diagnosis and on the general state of the patient, selects the best treatment.

Therefore, the tools that we use to make such decisions – in particular, AI/ML tools – must allow us:

- to provide an efficient and adequate description of the real-world situation and processing of the corresponding information: to reflect the actual physical dependence, not to lose any information, not to waste time processing irrelevant information, and to make sure that the resulting models adequately describe real-life constraints – both constraints on individual values and relations between pairs of values;
- to provide an efficient way to make a decision, and
- to make sure that the resulting decisions are understandable to us, human users.

In the following sections, we describe each of these criteria in more detail, and we show that for each of these optimality criteria, ReLU is the optimal activation function.

3 Which activation functions provide the most adequate description of the physical world

Which properties of the physical world are we talking about. Different real-world processes follow different laws. However, there are some features which are common to many such processes.

The world is described by differential equations. Starting with the successes of Newtonian mechanics, physical processes are described by systems of differential equations. The fact that the world is described by differential equations means that every change is gradual, it consists of several stages on each of which some small changes are performed – and the resulting big change is a *composition* of such smaller changes.

We also need constraints. Also, in many cases, in addition to equations, we also have additional inequalities that a physical system must satisfy. For example, differential equations of Newtonian physics remain valid when we change the order of time, i.e., replace time t with $-t$. However, from the practical viewpoint, this makes no sense.

Indeed, if I accidentally drop a cup and it breaks, I may be not happy about it, but this is in perfect accordance with physics. However, a time-reversed process, when the parts of the cup get together and form the original whole cup, does not make any physical sense. To avoid such non-physical processes, we need to impose a restriction from the Second Law of Thermodynamics: that the entropy at each moment of time should be greater than or equal to the entropies in the previous moments of time. Similar *inequalities* are needed in many other situations.

Which activations functions best capture the corresponding properties: composition property and inequalities property. It turns out that for capturing both composition property – corresponding to differential equations – and inequality property, the most adequate activation function is ReLU. Let us show it for each of these two properties.

ReLU is most adequate with respect to composition property. How can we use the fact that real dynamics usually comes as a composition of a large number of small steps? A similar fact is used in statistics; see, e.g., [70]. Namely, there is usually a lot of randomness and noise in many physical processes. Because of this noise, measurement results are, in general, different from the actual (unknown) value of the corresponding physical quantity. Usually, what we observe is a joint effect of many small independent random sources. If we add two independent random variables x and y of this type – each of which is a sum of many small independent random sources – then their sum z is also a sum of many small independent random variables. So, to have a good first approximation to such real-life random noises, it makes sense to find a few-parametric family of distributions for which the sum of two independent random variables from this family will also have a distribution from this family. It is known that under some reasonable conditions (such as scale- and shift-invariance), the smallest such family is the 2-parametric family of all possible normal (Gaussian) distributions. And indeed, under some natural conditions, the sum of many independent small random variables tends to normal distribution when the number of variables increases. This is known as the Central Limit Theorem; see, e.g., [70].

It makes sense to apply a similar approach to our case. In our case, each transformation is a composition of many close-to-identity transformations. In general, if we have transformations $f(x)$ and $g(x)$ each of which is a composition of many close-to-identity transformations, then their composition $f(g(x))$ is also a composition of many close-to-identity transformations. So, to have a good first approximation to such real-life transformations, it makes sense to find a few-parametric family of transformations for which the composition of two transformations from this family will also be a transformation from this family. In this case, it is possible to have a 0-parametric family, i.e., to have a family consisting of a single function $f(x)$ – with the property that the composition of this function with itself is equal to this same function: $f(f(x)) = f(x)$.

In many physical situations, the order between the value makes physical sense, so it is reasonable to restrict ourselves to functions that preserve this order, i.e., for which $x \le y$ implies that $f(x) \le f(y)$. It turns out that all the non-identity functions $f(x)$ which are monotonic and for which $f(f(x)) = f(x)$ are of the following three types: $f(x) = \max(a, x)$, $f(x) = \min(a, x)$, and $f(x) = \max(a, \min(b, x))$.

The first two functions are equivalent to ReLU in the sense that each network consisting of such neurons is equivalent to a ReLU network with the same number of neurons. Indeed, e.g., $\max(a, x) = \max(0, x-a)+a$. Since in a neural network, we already have linear transformations between any two applications

of a nonlinear function, we can just include $x \mapsto x - a$ and $y \mapsto y + a$ in these transformations. Similarly, $\min(a, x) = a - \max(0, a - x)$. The third function can be represented by two ReLU units corresponding to min and max; see [7] for technical details.

ReLU is most adequate with respect to inequalities. Suppose that we have a physical inequality $x \geq a$. To make related computations adequate, we need to make sure that on each computational step, our estimate for the quantity x satisfies this inequality. Computations deal with approximate data, algorithms are often approximate. So, we may end up with an approximate value that is somewhat smaller than the desired threshold a. In this case, we need to correct this estimate. For this purpose, we need to have a correction function $f(x)$ that will keep values $x \geq a$ intact, but that will transform values smaller than a into physically possible values – i.e., into values $\geq a$.

In many physical situations, the order between the values makes physical sense, so we want this transformation to preserve this order: if $x \leq y$ then we should have $f(x) \leq f(y)$. It turns out (see, e.g., [20]) that the only correction function that satisfies this monotonicity property is the function $f(x) = \max(a, x)$, i.e., a function which – as we have shown – is equivalent to ReLU.

4 Which activation functions do not lose information and do not process irrelevant information

As planned, we will discuss which activation functions best preserve constraints on individual variables, and which best preserve relations between variables.

4.1 Which activation functions best preserve constraints on individual variables

To analyze which activation functions do not lose information and do not process irrelevant information, let us first consider this problem locally, on the level of a single input z to the activation function $s(z)$. In other words, let us analyze when the transformation from z to $y = s(z)$ does not lose information and, at the same time, does not add irrelevant information. The details of this analysis can be found in [21].

To provide this analysis, let us take into account that usually, the inputs come from measurements, and measurements are never absolutely accurate, there is always some measurement uncertainty. For example, if we measure distance with accuracy 1 cm, then, in effect, possible values of the result are 0 cm, 1 cm, 2 cm, etc. – every other reading of a measuring instrument is indistinguishable from one of these values. In general, if we denote the measurement accuracy by h, this means, in effect, that we only consider values proportional

to h, i.e., values 0, h, $2h$, etc. For such a measurement, on an interval of width w, we have w/h distinguishable values.

If after the transformation $z \mapsto s(z)$, we have fewer values, this means that some previously distinguishable values will stop being distinguishable – i.e., we will lose some information.

Similarly, if after the transformation, we will have more values than before, this means that some values that are not distinguishable based on the available information are now claimed to be distinguishable. In other words, in this case, we add some distinction that is not related to the available information – and this is exactly what we wanted to avoid, time-wasting processing of irrelevant information.

So, the only case when we do not lose any information and, at the same time, do not waste time processing irrelevant information, is when on each interval, the number of distinguishable values does not change after this transformation. Since the number of distinguishable values is equal to the interval width divided by the accuracy h, this means that the transformation must preserve the interval width.

Of course, measurement accuracy depends on the measured value: e.g., when we measure distances in a small room, we can easily reach a 1 cm accuracy, while reaching this accuracy for a kilometer distance requires much more accurate measuring instruments. Thus, the above conclusion is only applicable to a small vicinity of each value, on which we can safely assume that the accuracy remain the same for all the values from this interval. For a narrow interval $[z, z+w]$, a smooth function is usually monotonic – unless we happen to encounter one of the rare points where the derivative $s'(z)$ is equal to 0. For a monotonic function, values from the interval $[z, z+w]$ are transformed into values from the interval with endpoints $s(z)$ and $s(z+w)$. Thus, the width of the transformed interval is equal to $|s(z+w) - s(z)|$. For small w, we have $s(z+w) \approx s(z) + w \cdot s'(z)$, so the width of transformed interval is equal to $w \cdot |s'(z)|$. Thus the requirement that the width should not change after the transformation means that we should have $w \cdot |s'(z)| = w$, i.e., $|s'(z)| = 1$. Thus, for each z, we should have either $s'(z) = 1$ or $s'(z) = -1$. Hence, the activation function $s(z)$ should consist of parts on which $s'(z) = 1$ – and thus, $s(z) = z + \text{const}$, and parts with $s'(z) = -1$ – and thus, $s(z) = -z + \text{const}$.

We cannot have just one part – because in this case, the function $s(z)$ will be linear, and the resulting neural network will only be able to compute linear function, while many real-life dependencies are nonlinear. Thus, we must have at least two parts. The simplest case is when we have exactly two parts, e.g., when we have $s(z) = |z|$. One can show that every such activation function is equivalent to ReLU – in the same sense as before, that every function computable by a neural network with such an activation function can be computed by a ReLU-based network with the same number of neurons – and vice versa. For example, for $s(z) = |z|$, ReLU can be described as $0.5 \cdot (s(z) + z)$. So, a ReLU neuron can be simulated by a $|z|$-type neuron, with linear transformation after this neuron.

Comment. We can formulate the same property in a more commonsense-type form: if a is close to b, then $s(a)$ should be close to $s(b)$, and vice versa: if $s(a)$ is close to $s(b)$, then a should be close to b. It turns out that if we use fuzzy logic (see, e.g., [12, 34, 60, 64, 65, 78]) to formalize this requirement, then we get exactly the same property – that $|s(a) - s(b)| = |a - b|$ when a and b are close – which leads to the same conclusion [75].

4.2 Which activation functions best preserve relations between variables: scale-invariance

Even when we have a perfect representation of each input, we may still process irrelevant information if there is a relation between the inputs, One such case is the fact that while we want to process the values of physical quantities, what we really process is their numerical values, and the numerical values change if we change a measuring unit. For example, when we go from meters to centimeters, all numerical values are multiplied by 100. In general, if we replace the original unit by a new unit which is $c > 0$ times smaller, then all numerical values are multiplied by c.

It therefore makes sense to require that the results of a neuron processing should not depend on which units we use. If the neural network produces different results depending on what units we use, this means that this network processes irrelevant information – namely, the information about what measuring unit was used. So, if, in the original units, we had $y = s(z)$, then in the new units we should have $Y = s(Z)$, where $Y = c \cdot y$ and $Z = c \cdot z$ are the values in the new units.

Substituting the expressions for Y and Z into the formula $Y = s(Z)$, we conclude that $c \cdot y = s(c \cdot x)$. Since y is equal to $s(z)$, we conclude that $s(c \cdot z) = c \cdot s(z)$ for all z and for all $c > 0$. In particular, for any value $x > 0$, we can take $c = x$, $z = 1$, and get $s(x) = c_+ \cdot x$, where we denoted $c_+ \stackrel{\text{def}}{=} s(1)$. Similarly, for any value $x < 0$, we can take $c = -x$, $z = -1$, and get $s(x) = c_- \cdot x$, where we denoted $c_- \stackrel{\text{def}}{=} -s(-1)$.

Thus, the only activation function that satisfies this scaling-invariance property is a piecewise-linear function of the above type – and we have already mentioned that all such functions are equivalent to ReLU; see, e.g., [39, 45, 46, 69]. This provides more explanation of why ReLU is so effective – it is the only activation function that does not use the irrelevant information about measuring units.

Comment. Selecting a different measuring unit is only one of the transformations that changes the numerical values while keeping physics intact. In the following text, we will list several other transformations of this type.

- For example, we can change the starting point from the original 0 value to a new 0 value which is a units before. In this case, all numerical values are shifted, from x to $x + a$.

- To describe points in 3D space, we can use different coordinate systems. In a different system, each point will be characterized by different numerical values of its coordinates, but it will be the same point.
- Sometimes, there are also nonlinear transformations of this type.

5 Which activation functions provide the most efficient data processing

Let us describe this objective in precise terms. While modern computers are very fast, there are still many practical problems for which even faster computations are needed. For example, at present, computer-based systems reasonably accurately predict tomorrow's weather – so accurately that days when prediction was wrong are pretty rare. A few hours of computer time on a high-performance computer – and a 24-hour prediction is ready.

These predictions work well when we have usual weather, they also work reasonably well when we have storms and hurricanes. But there is one weather phenomenon for which predictions are still not realistically possible: tornadoes. Every year, tornadoes bring in a lot of damage and even loss of lives. So why cannot we predict their behavior? In principle, tornadoes are an atmospheric phenomena, they are described by the same Navier-Stokes equation as weather – and indeed, we can, in principle, "predict" the tornado's path. We placed the word "predict" in quotes because tornadoes move and change much faster than other meteorological phenomena: their change in 15 minutes is similar to the daily change in normal weather. As a result, predicting where the tornado will go in the next 15 minutes takes the same amount of computational effort as predicting tomorrow's weather – several hours on a high-performance computer. For predicting tomorrow's weather, such a prediction makes perfect sense, but for a tornado, it is useless: the tornado will turn somewhere in 15 minutes without waiting for our computations to end.

This is just one of the examples, there are many other examples of real-time decision making where faster computations are desirable. From this viewpoint, what we want is to be able to compute as fast as possible – and this means, in particular, that we want to select an activation function that is as fast to compute as possible.

So, from this viewpoint, we want to select an activation function $s(z)$ that is the fastest to compute.

Which activation function is the fastest to compute? In a computer, the fastest operations are the ones that are directly hardware supported. In most computers, the only directly hardware supported operations are arithmetic operations: min, max, addition/subtraction, multiplication, and division. Also, shifts – which are equivalent to multiplying by a power of 2 – are easy to compute.

Whatever we want the computer to compute is actually implemented as a sequence of such elementary operations. For example, when we ask the com-

puter to compute $\exp(x)$, it actually computes the polynomial approximation to $\exp(x)$ – that is equal to the sum of the first few terms in this function's Taylor expansion.

Out of these operations, the fastest are min and max, since these operations do not involve much data processing: they simply compare the two numbers and select one of them. Next fastest are addition/subtraction. Multiplication includes several additions – just like when we do it by hand – so it is somewhat slower. Similarly, division contains several multiplications – also similarly to how we do it by hand – so it is even slower.

From this viewpoint, the fastest-to-compute activation function $s(z)$ is the one that uses only min or max – of the input z and possible some constant c. The easiest-to-implement constant is usually 0, so we end up with $\max(0, z)$ and $\min(0, z)$. The first of these expressions is exactly ReLU, the second is equivalent to ReLU – if we take into account that between several application of the activation function, there are linear transformations. In our case, $\max(0, z) = -\min(0, -z)$. So, if we first apply a linear operation $z \mapsto -z$, then apply the $\min(0, z)$ activation function, and then again change the sign, we get exactly ReLU. Thus, whatever we can compute with ReLU neurons can be computed with the exact same number of neurons with activation function $\min(0, z)$. In other words, ReLU (and operations equivalent to ReLU) is optimal with respect to this optimality criterion – i.e., in this case, it is the fastest to compute.

Comments.

- For technical details, see [48, 77].
- It is worth mentioning that selecting the simplest possible algorithm not only leads to the fastest computations, it has an additional advantage: namely, it has been proven that if we always select the simplest possible model describing current observations, then eventually, we will converge to the actual description of the physical world; see, e.g., [56].
- A related alternative explanation for ReLU comes from the arguments presented in [38, 44]. It is based on the fact that data processing is, in some sense, similar to chemical – or other – reactions: we combine data points and get new results, just like we combine substances and get new substances. From this viewpoint, it is desirable to simulate the fastest possible chemical reactions. Most chemical reactions are well described by equations of chemical kinetics, in which the reaction rate is proportional to the product of the concentrations $a, b, \ldots$ of different substances, e.g., to $a \cdot b$ if we have two substances. This law has a simple interpretation: molecules are randomly distributed in space, and the reaction rate is proportional to the probability that molecules of two type happen to be at the same location. However, for fast reactions – e.g., reactions on the surface of a catalyst, where concentrations are high – molecules do not need to find each other, they are all already close. In this case, each

molecule of a immediately starts reacting with a molecule of b – as long as there are pairs of such molecules. Thus, the reaction rate is proportion to the amount of such pairs, i.e., to $\min(a, b)$. Since fast chemical reactions use the operation $\min(a, b)$, then the idea of simulating fast physical processes leads to the use of this operation in data processing as well. The result of this operation can be equivalently described as $a - \max(0, a - b)$. Thus, a network consisting of such operations is equivalent to a network of ReLU neurons.

6 Which activation functions are most appropriate for decision making

Why ReLU is the most appropriate for decision making: our main explanation. As we have mentioned, the ultimate goal of all data processing is to make decisions, and ideally, to make optimal decisions, i.e., decisions that minimize some objective function. In some practical situations, we want to maximize the objective function $f(x)$ – but from the mathematical viewpoint, such problem cane be reduced to minimization of the objective function $g(x) \stackrel{\text{def}}{=} -f(x)$.

In general, minimization is an NP-hard problem, but there is a class of problems for which minimization is feasible – these are problems with convex objective functions. It can be proved that convex functions are the largest of such class – if we add at least one non-convex function and allow all possible convex combinations, the minimization problem for this enlarged class becomes NP-hard [33].

Convex minimization problems are ubiquitous in practice. So, to preserve efficiency, it makes sense to select an activation function for which the neural approximation to a convex dependence retains as many convexity properties as possible. Thus, it is desirable to require that the activation function itself is convex, and that every other convex function of one variable – that is approximated by a linear combination of shifted and re-scaled activation functions – should be represented as a convex combination as such functions.

The set S of all convex functions is itself convex – in the sense that a convex combination $c_1 \cdot f_1(x) + \ldots$ of convex functions is also convex. If the activation function is in the interior of this set S, then its linear combinations will not cover convex functions on the border of this set. So, the activation function must be one of the extreme points of this set. According to calculus, a function $y(x)$ is convex if and only if its second derivative $y''(x)$ is everywhere non-negative. The extreme points of this set are, thus, functions for which $y''(x) = 0$ everywhere except for one value x_0. If we integrate this expression, we get a piece-wise constant function $y'(x)$, and if we integrate it again, we get a piece-wise linear function with two linear parts – and we already know that from the viewpoint of a neural network, all such functions are equivalent to ReLU. Thus, ReLU is indeed the most appropriate for decision making; see [17] for details.

A related second explanation. The paper [17] also has a related alternative explanation of why ReLU is effective in decision making. This explanation is as follows. We want to select the most appropriate activation function from the set S of all convex functions. In precise terms, we select some objective functional $J(s)$ that describes the appropriateness of each objective function s, and we want to select a function s for which this appropriateness $J(s)$ attains its largest possible value.

According to calculus, a function attains its maximum on a bounded domain D at some point x if either all its partial derivatives at x are equal to 0 or this value is located on the border of the domain. When the domain is small, the probability that is contains one the few points x where all partial derivatives are zeros is very small, so the maximum is almost always attained on the border. We can apply the same argument to the function limited to the border – and eventually conclude that the maximum is most probably obtained at the extreme points of the domain D – e.g., if the domain is a polytope, maximum is most probably obtained at one of its vertices.

In our case, the domain is the set of all convex functions, and we already know that the extreme points of this domain are exactly piece-wise linear functions equivalent to ReLU. Thus, we get another explanation of why empirically ReLU is the most appropriate activation function for decision making.

Comment. An additional decision-related argument – provided in [22] – explains why non-smooth activations functions such as ReLU work better while most physical dependencies are smooth. This explanation is based on the fact that the ultimate goal of science and engineering in general is to make decisions, and decisions means that we switch from one option to another – i.e., decision is a function that is not only not smooth, it is not even continuous. Not surprisingly, to describe decisions, non-smooth functions are more appropriate than smooth ones.

Specifically, [22] shows that the simplest rule of group decision making – the majority rule – can be easily computed by a simple 2-ReLU-neuron network, no matter how many participants n we have, while if we use neurons with smooth activation functions, we will need exponentially many neurons.

7 Which activation functions lead to most understandable results?

One of the natural requirements is that the results of a neural network should be as understandable as possible. Ideally, not only the results of the neural network should be understandable, but also each step in the computations should be understandable. In other words, the result $s(w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n)$ of a processing by each neuron should be understandable.

Understandable means, in particular, that this results should be logical, i.e., that we should be able to describe it in terms of the usual logical connectives "and", "or", and "not". Of course, if we are talking about decisions, after the

very first processing of a single neuron, we do not yet get the final decision, but we may get some degree of confidence for or against some possible decisions. From this viewpoint, it is reasonable to view the signals as values as such degrees – this is exactly what is done when we use a neural network to make a classification decision.

So, in our interpretation, the inputs to the corresponding "and"- and "or"-operations should be not the usual binary truth values, but rather our degrees of confidence. In other words, we need to extend the usual logical operations from the set $\{0, 1\}$ of binary truth values to a continuous domain – e.g., to the interval $[0, 1]$. Such extensions are well-known and well-studied in expert systems and fuzzy logic (see, e.g., [12, 34, 60, 64, 65, 78]). These operations must satisfy some natural properties. For example, since $A \,\&\, B$ and $B \,\&\, A$ mean the same thing, then for the corresponding "and"-operation $f_{\&}(a, b)$, we should have $f_{\&}(a, b) = f_{\&}(b, a)$ for the extended operations. Similarly, since for false statement A, the composite statement $A \,\&\, B$ is also clearly false, we should have $f_{\&}(0, b) = 0$ for all b. Since for a true statement A, our degree of confidence in the statement $A \,\&\, B$ is exactly the same as our degree of confidence in B, we must have $f_{\&}(1, b) = b$ for all b.

Similarly, for the "or"-operation, we should have $f_{\vee}(a, b) = f_{\vee}(b, a)$ for all a and b, and $f_{\vee}(0, b) = b$ and $f_{\&}(1, b) = 1$ for all b. It turned out that the only activations functions $s(z)$ for which a single neuron can represent an "and"-operation are the ones that are equivalent to ReLU $s_0(z)$ – in the same sense as before, i.e., that for some coefficients a_i, we have

$$s(z) = a_0 + a_1 \cdot s_0(a_2 + a_3 \cdot z) + a_4 \cdot z$$

for all z. Similarly, the only activations functions $s(z)$ for which a single neuron can represent an "or"-operation are the ones that are equivalent to ReLU; see [2] for proofs; see also [6, 8, 9].

From this viewpoint, ReLU indeed provides the maximum possible understandability.

8 How can we explain that different criteria lead to the same family of activation functions?

Our explanation of this fact is that all reasonable optimality criteria are scale-invariant, and for every scale-invariant optimality criterion, the optimal activation function is also scale-invariant – and thus, equivalent to ReLU. Let us describe this explanation in detail.

8.1 What do we mean by optimal?

What is optimal: naive idea and its limitations. Our goal is describe what is optimal. We therefore need to formally describe what we mean by optimal.

At first glance, this sounds straightforward: we select an objective function $J(x)$, and we try to find the alternative x for which this objective function attains its largest possible value (or sometimes smallest possible value). However, it is easy to see that this description does not cover all the cases when we talk about optimality.

For example, if we are going to a conference that is not happening in a hotel, then a reasonable idea is to look for a hotel which is the closest to the conference site. In this case, the distance is the objective function. But what if we have two (or more) hotel at the same shortest distance? To choose between these two hotels, we need to apply some other criterion: e.g., select the cheapest if our funding is limited, or the one with the highest level of customer satisfaction of the room rate is not an issue.

Similarly, if we have two algorithms whose worst-case computation time is the same, we can choose, among them, the one with the smallest average computation time.

If this additional criterion does not lead to a unique solution, we can use the resulting non-uniqueness to optimize something else, etc. Let us see how we can translate these limitations into a general definition of optimality.

Towards a formal definition. To define optimality, all we need is to be able to decide, for all (or at least for some) pairs of alternatives which one is better (or that they are both of the same quality). These conclusions should be consistent: if a is better than b and b is better than c, then a should be better than c.

Let us denote the statement "a is better than b" by $a \succ b$, the statement "a and b are of the same quality" by $a \sim b$, and the statement "either a is better then b or they are of the same quality" by $a \succeq b$. Once we know the relation $\succeq$, we can determine the other two relations we well:

- the relation $a \sim b$ means that $a \succeq b$ and $b \succeq a$;
- the relation $a \succ b$ means that $a \succeq b$ and $b \not\succeq a$.

In these terms, the above consistency means that if $a \succeq b$ and $b \succeq c$, then $a \succeq c$.

We also always have $a \succeq a$. In mathematics, such relations are known as *pre-order* relations.

Comment. Partial orders are a particular cases of pre-orders that satisfy an additional condition of anti-symmetry: if $a \preceq b$ and $b \preceq a$, then $a = b$.

Towards a formal definition (cont-d). We should also require that there is only one alternative that is optimal with respect to this criterion: otherwise, as we have just mentioned, the current criterion would not be final: we could use the resulting non-uniqueness to optimize something else.

So, we arrive at the following definition.

Definition 1. *Let A be a set. Its elements will be called* alternatives.

- *By an* optimality criterion*, we mean a transitive (pre-order) relation $\succeq$ on the set A.*

- *We say that an alternative* $a_{\rm opt}$ *is* optimal *if* $a_{\rm opt} \succeq a$ *for all* $a \in A$.
- *We say that the optimality criterion is* final *if there is exactly one optimal alternative.*

8.2 Symmetries and invariances: a general reminder

What are symmetries. From the commonsense viewpoint, when can we make a prediction? If we encounter a completely new situations, it is very difficult to predict what will happen. But if we remember what happened in a similar situation(s) in the past, it is natural to conclude that a similar result will happen now.

If in the past, a person ate peanuts and got allergic reaction, it is natural to expect that if this person eats peanuts again, he/she will get allergic again. If we drop a ball and it starts falling down with an acceleration of 9.81 m/sec^2, then we expect the same thing to happen at a future date in a different Earth location, maybe at a different orientation of the ball.

If we see that Ohm's law is valid in Denmark (where it was originally discovered), and in many other countries, we conclude that it is valid everywhere.

The important word here is "similar". It means that there are some transformation that do not affect the results. For the ball drop, these transformation included shifts in time and space and rotations. In other cases, there may be other transformations: e.g., changing the signs of all electric charges from plus to minus and vice versa, making a smaller-size copy of a plane to test its aerodynamic properties, etc.

In physics, transformations that preserve some physical properties are called *invariance* or *symmetries*. Symmetries are one of the main tools in physics; see, e.g., [26, 72].

Mathematical comment. If a transformation from A to B preserves some properties, then naturally an inverse transformation also preserves the corresponding properties. For example, if, in general, nothing changes when we shift an object 1 m to the right, this implies that nothing changes when we shift the object back, 1 m to the left.

Similarly, in nothing changes when we shift an object 1 m to the right, and nothing changes when we rotate it 90 degrees, then nothing changes when we first shift and then rotate, i.e., in mathematical terms, when we apply a *superposition* (also known as *composition*) of these two transformations.

Thus, the set of all transformation should include, for two transformations $f(x)$ and $g(x)$, the inverse $f^{-1}(x)$ and the composition $f(g(x))$. Sets with this property are known as *transformation groups*.

Comment. It is worth mentioning that composition is when the output of a function – in the above example, $z = g(x)$ – serves as input to another function – in the above example, $y = f(z)$. This is what happens when we perform several computational procedures one after another. This is also what happens

in a neural networks (in particular, in a deep neural network), where outputs of some neurons serve as inputs to other neurons.

How are symmetries related to optimality? By definition, invariances do not change important properties of the situation. Thus, it makes sense to conclude that if a is better than (or of the same quality as) b, then after a symmetry $f : A \mapsto A$, we will still have $f(a) \succeq f(b)$. Thus, we arrive at the following definition.

Definition 2.

- *Let $f : A \mapsto A$ be a 1-to-1 mapping. We say that an optimality criterion $\succeq$ is* f-invariant *if for every pairs a and b, we have $a \succeq b$ if and only if $f(a) \succeq f(b)$.*
- *Let F be a family of 1-1 functions from the set A to itself. We say that an optimality criterion $\succeq$ is* F-invariant *if its f-invariant for all $f \in F$.*

We are almost ready to formulate our result. To do that, we need to formulate one more definition.

Definition 3.

- *Let $f : A \mapsto A$ be a 1-to-1 mapping. We say that an alternative $a \in A$ is* f-invariant *if $f(a) = a$.*
- *Let F be a family of 1-1 functions from the set A to itself. We say that an alternative a is* F-invariant *if its f-invariant for all $f \in F$.*

Proposition. [69] *For every final F-invariant optimality criterion, the optimal alternative is also F-invariant.*

Proof of the Proposition. By definition of the final optimality criterion, there exists the optimal alternative a_{opt}, i.e., the alternative for which $a_{\text{opt}} \succeq a$ for all $a \in A$. In particular, for every $f \in F$, we have $a_{\text{opt}} \succeq f^{-1}(a)$.

Since the optimality criterion $\succeq$ is F-invariant, from $a_{\text{opt}} \succeq f^{-1}(a)$, we can conclude that $f(a_{\text{opt}}) \succeq f(f^{-1}(a))$. Here, by definition of the inverse function, $f(f^{-1}(a)) = a$. Thus, we have $f(a_{\text{opt}}) \succeq a$ for all $a \in A$.

By definition of the optimal alternative, this means that the alternative $f(a_{\text{opt}})$ is optimal. However, since the optimality criterion is final, there exists only one optimal alternative. Thus, indeed, $f(a_{\text{opt}}) = a_{\text{opt}}$ for all $f \in F$, i.e., indeed, the optimal alternative is F-invariant. The proposition is proven.

8.3 The resulting explanation

In our case, it makes sense to require that the optimality criterion is invariant with respect to scaling $x \mapsto \lambda \cdot x$: this may be computational simplicity, this may be interpretability, all of them should not change if we use another unit for measuring inputs. In this case, due to the above Proposition, the optimal activation function should also be scale-invariant, and we have already mentioned that the only scale-invariant activation function is ReLU.

9 Other empirically successful activation functions can be similarly explained: in brief

While ReLU activation function is, *in general*, the best, other activation functions are also effectively used in different *specific* situations. For many of these non-ReLU activation functions, similar arguments explains their empirical success:

- *exponential* linear units $s(z) = \alpha \cdot (\exp(\beta \cdot z) - 1)$; their empirical success can be explained by scaling [24];
- *fractional-linear* activation functions $s(z) = (1 + b \cdot z)/(c + d \cdot z)$; their empirical success can be explained by the fact that they are the easiest to compute – after ReLU, of course [54];
- *leaky* ReLU functions, for which $(z) = z$ for $z > 0$ and $s(z) = \alpha \cdot z$ for $z < 0$; their empirical success can be explained by scaling [24];
- *polynomial* and other activations functions; their success can also be explained by scaling [52];
- *Rectified Power* (RePU) activation functions $s(z) = \max(0, z^a)$; their success can be explained by scale-invariance [14];
- *sigmoid* activation function $s(z) = 1/(1 + \exp(-z))$ can be explained by invariance with respect to nonlinear rescalings [39, 45, 71]; in particular, in [71], this invariance is used to show that these functions are the least sensitive to additive noise;
- *spiking* neural networks, with an activation function that is different from 0 only in a single point – their empirical success can be explained by scale-invariance [11];
- *squashing* functions

$$s(z) = \frac{1}{\lambda \cdot \beta} \cdot \ln \frac{1 + \exp(\beta \cdot z - (a - \lambda/2))}{1 + \exp(\beta \cdot z - (a + \lambda/2))};$$

  their empirical success can also be explained by scale-invariance [23, 74];
- *wavelet* activation functions are explained in [53],

10 Other features of machine learning can be similarly explained: a brief overview

10.1 What are these features

In the previous sections, we concentrated on the explanations of the empirical success of different activation functions. In this section, we briefly overview

results showing that other empirically successful features of machine learning can be similarly explained. The explanations of the selections for these features are listed in the order in which they need to be selected:

- first, we need to decide which machine learning tool to use, whether it is neural networks or some other tool;
- if we select neural networks, we need to decide how many hidden layers we should use, i.e., whether we want to use shallow (one-hidden-layer) or deep (multiple-hidden-layers) neural networks;
- if we select a deep neural network, we need to decide on its architecture, e.g., whether inputs to each layer should only come from the directly preceding layer or from the other previous layers as well;
- we need to decide whether we need some data pre-processing – and which exactly;
- to train the resulting neural network, we need to decide on the objective function, on the initial values of the weights, and on the techniques for optimizing the objective function – and we need to explain why these selections are effective;
- if we repeat the procedure several times – to get more accurate results – we need to decide on the best way to combine different results;
- finally, if our ultimate goal is to make a decision – as if often the case – how can we make a decision based on the neural network results.

As we mentioned earlier, there is too much related material to fit into a single paper. So, in this section, we only provide very brief overviews of different explanations with links to details – and in some cases, we only provide the links.

10.2 Which machine learning tool to use

In many applications, neural networks are, at present, the most effective machine learning tool. But why?

A usual – somewhat naive explanation – is not very convincing. In mathematical terms, as we have mentioned earlier, using a neural network to represent a dependence means representing this dependence as a composition of functions $s(w_0+w_1\cdot x_1+\ldots+w_n\cdot x_n)$ describing an individual neuron. For many activation functions $s(z)$, such compositions are universal approximations. This is sometimes used as an explanation for why neural networks are empirically successful.

However, we do not find this explanation very convincing: there are many other families that have the universal approximation property, e.g., polynomials of Fourier series. So why are neural networks more empirically successful than other families?

We have two general explanations for this, and one specific explanation – of why neural networks are often more effective than Support Vector Machines.

Our first explanation: neural networks lead to fastest computations. Our first explanation (see, e.g., [10, 42, 46, 51]) is related to the need for computation speed. The faster the elementary computational steps, the faster the overall computations – especially since many computations can be parallelized, in which the overall time is equal to the longest time of each parallelized computations. We previously mentioned that:

- the fastest computational operations are min and max, but they do not lead to any new values, while
- the next simplest are addition/subtraction and shift, while lead to generic linear combinations.

So, the fastest-to-compute functions are linear functions.

However, if we only compute linear functions, we will only get linear functions – since the composition of linear functions is also linear. And it is well known that some real-life dependencies are nonlinear. Thus, in addition to linear computational units, we also need computational units computing nonlinear functions. We also mentioned earlier that the more inputs, the longer computations. Thus, the fastest nonlinear units are the ones that compute a function of one variable $y = s(z)$.

Outputs of some computational units can serve as inputs to others. It makes no sense to feed outputs of linear units into a new linear unit – the resulting functions – composition of two linear functions – is also linear, so it could be computed twice faster by a single linear unit instead of two. Similarly, it makes no sense to feed the output $y = s(z)$ of a nonlinear unit into a new nonlinear unit $t = s_1(y)$: the resulting function – composition $s_1(s(z))$ of two functions of one variable – is also a function of one variable, so it could be computed twice faster by a single nonlinear unit instead of two.

Thus, in the fastest-to-compute arrangement, the output $z = w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n$ is fed to a nonlinear unit $y - s(z)$, resulting in a signal $y = s(w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n)$ – which is exactly what a neuron does. Thus, the need to make computations as fast as possible indeed leads to neural networks.

Our second explanation – related to invariance. Another explanation [16] is based on the fact that often, the representations of the input as n values $x_1, \ldots, x_n$ is rather arbitrary. For example, we can represent a point in a 3-D space by its coordinates, but we can select different coordinate systems. Usually, we select linearly related coordinate systems, so that the transformation from one coordinate system to another is linear. It therefore makes sense, in selecting an approximating family, to select an optimality criterion that would not change if we apply linear transformation to the inputs. It turns out that for every such criterion, the optimal family indeed corresponds to neural networks.

Why, in most cases, deep neural networks are more efficient than Support Vector Machines (SVMs). At first glance, SVMs should be more efficient. Indeed, in SVM, we approximate the empirical dependence by a linear combination of several known functions. So, the match between each data point and the approximating function is described by a linear equation – and efficient algorithms are known for solving systems of such equations. In contrast, in neural networks, the approximating function depends nonlinearly on parameters. So, to find these parameters, we need to solve a system of nonlinear equations – and solving such systems is, in general, NP-hard.

An explanation is given in [13]: if all inputs were independent, then yes, SVMs would be better. However, in practice, there are many strong dependencies between the inputs – our Universe is not lawless, there are many laws regulating it. As a result, the number of truly independent inputs is much smaller than the overall number of inputs – and deep neural networks capture this perfectly, since the number of neurons in the first layer is smaller than the overall number of inputs; see a more detailed description in the next subsection. So, deep neural networks combine inputs into fewer ones and later deal with these fewer inputs. This explained why they are often more efficient than SVMs – since SVMs do not take this dependence into account. A similar argument explains why sparse techniques – techniques that take into account that the desired outputs can be reasonably accurately described as function of only a few of thhe inputs – are so efficient in practical signal and data processing.

Sometimes, other techniques are better. While in general, neural networks are better, in many practical situations, other techniques work better. It is therefore desirable to understand when other techniques are better – and why.

For example, it has been observed that neural networks do not always perform well – compared to other methods – in NP-hard problems. The following explanation for this phenomenon was given in [36]. Current machine learning technique is based on gradient descent, and gradient descent works well in problem in which the objective function has only one global minimum and no local minima. When a problem has several local minima, then in situations when the starting point is close to a local minimum, gradient descent will lead to this local minimum. NP-hard problems usually have several local minima – otherwise, we would be able to solve them by a straightforward gradient descent algorithm and they would, therefore, be tractable. This explains why for NP-hard problems, deep learning does not always work so well.

For NP-hard problems, many other machine learning techniques can be successfully used. Specifically, for fuzzy-based learning techniques, the paper [47] analyzes the question of when neural techniques are better and when fuzzy techniques are better.

10.3 Deep or shallow neural networks?

Already for a single hidden layer, neural networks have a universal approximation property, so why do we need several layers? Why are deep networks – with

several layers – more empirically successful?

Our first explanation: deep networks lead to more accurate results. From the viewpoint of speed, shallow neural networks, with only one non-linear layer, are the fastest, so this is what needs to bs used on very time-critical problems. However, in many other problems, it is also important to make accurate predictions and reliable decisions. For this purpose, the neural networks should approximate the actual dependence as accurately as possible.

For this, we need to have as many possible models as possible – within the given budget of bits B. In general, by using B bits, we can describe 2^B different models. However, when we have K neurons in the same layer, then any permutation of these neurons leads to the same model. Since there are $K! \stackrel{\text{def}}{=} 1 \cdot 2 \cdot \ldots \cdot K$ possible permutations, the number of different models decreases to $2^B/K!$. To increase the number of possible models – and, to make them more accurate – a natural idea is to decrease the number of neurons K in a single layer, i.e., to place these neurons in several layers; such networks, with several nonplinear layers, are what is called deep [39, 46, 45].

Our second explanation: if we take into account that inputs come with uncertainty, then only deep networks have universal approximation property. A related explanation – also based on approximation accuracy – is given in [4, 57] (based on [52]). This explanation takes into account that whether we implement the activation function in hardware or in software, its value is computed only approximately, with some approximation accuracy ε. Within this accuracy, on the domain bounded by the smallest and largest values of all the input, we can approximate $s(z)$ with this accuracy, by a polynomial of some degree n. Thus, if we use a single hidden layer – i.e., if we return a linear combination to the neuron's output as the computation result – then, in effect, all we compute are polynomials of the same degree n, and the class of all such polynomials does not have the universal approximation property. In contrast, if we use several layers, we can get compositions of such polynomials – which can be of degrees n^2, n^3, etc.

This result also shows that to get the universal approximation property in this realistic case – when we take into account that computations are not perfectly accurate – we cannot limit ourselves to any fixed number of layers – such a limitation will limit the power of the corresponding polynomials and thus, prevent such networks from being universal approximators.

Why sometimes shallow neural networks work better. While in general, deep learning is more effective, sometimes, more traditional – shallow – neural networks, with only one hidden layer, work better. It is therefore desirable to be able to predict when shallow networks will work.

In some cases, the problem is so time-critical that at this moments, we need to use the fastest possible computational device to get the result – and, as we have mentioned earlier, shallow neural networks are the fastest. Sometimes, however, shallow networks work better even if we have extra time. So why is it so, and how can we predict when shallow networks work better?

A partial answer to this question is given in [67]. Namely, from the mathematical viewpoint, a function computed by a multi-layer neural network is a composition of several nonlinear functions – corresponding to different layers. So, it makes sense to conclude that such computations are appropriate when the dependencies that we are trying to describe can be naturally described as such compositions. In other words, they are appropriate if we have a class of functions that is closed under composition – i.e., a class that includes, with every two functions, their composition as well. To describe the reverse dependencies – the dependence of x on y as opposed to the dependence of y on x – it makes sense to also assume that this class contains, with each function, its inverse. In mathematics, classes of transformations that are closed under composition and inverse are known as *transformation groups*. They are ubiquitous in physics, where they describe natural invariances known as *symmetries* [26, 72] – e.g., most physical processes do not change if we simply shift and/or rotate the whole the whole system, in which case rotations and shifts form such a group. Thus, we can conclude that deep learning is more appropriate if the problem has natural symmetries – while in situations where there is no such natural symmetry, shallow networks may work better.

This argument sounds somewhat too philosophical, but a simple example confirms this conclusion: if we consider the simplest nonlinear smooth activation functions – namely, quadratic ones – then the simplest way of combining two such neurons to compute a 4th order polynomial of one variable is possible if and only if this polynomial is invariant with respect to a reflection $x \mapsto a - (x - a)$ with respect to some point a.

10.4 Selecting the best architecture

The empirical success of *residual* neural networks, when neurons can get inputs not only from the directly preceding layer, but also from all preceding layers, is explained in [31].

10.5 Pre-processing: why max-pooling?

Need for preprocessing. In shallow neural networks, we could have any number of neurons in the first layer, and thus, we could deal with any number of inputs. However, as we have mentioned, we can reach better accuracy if we place these neurons in several layers. In this case, we cannot process as many inputs as, e.g., a typical image has, we need to compress it.

Which pre-processing techniques are empirically successful. Empirical tests shows that the most effective compression is achieved when we use either max or arithmetic average of neighboring values.

How can we explain this success. Similar to the above explanations for the empirical success of ReLU activation functions, we have two related explanations for the empirical success of max-pooling:

- a scaling-based explanation for this empirical phenomenon is given in [25, 39, 45];
- an explanation based on the desire to have fastest pooling operations is given in [77].

Comment. Usually, we pull together values at the neighboring points. However, in some cases, it is more convenient to pull together values in non-neighboring points – namely, in points from an appropriate sub-grid of the usual grid of pixels. This is known as *dilated* Convolutional Neural Network. Empirical success of such networks can be explained by shift-invariance [19].

10.6 What objective function should we use

In general, in machine learning, we have the outputs y_k corresponding to several input tuples x_k, and we want to find the parameters c of the corresponding computational model $f(x, c)$ for which $z_k \stackrel{\text{def}}{=} f(x_k, c)$ is close to y_k for all k. From a commonsense viewpoint, this means that we want the tuple $(z_1, z_2, \ldots)$ to be as close to the given tuple $(y_1, y_2, \ldots)$ as possible. Both tuples are elements of the multi-dimensional space, so it is reasonable to minimize the usual Euclidean distance in this space $\sqrt{(z_1 - y_1)^2 + (z_2 - y_2)^2 + \ldots}$ – which is equivalent to minimizing its square $(z_1 - y_1)^2 + (z_2 - y_2)^2 + \ldots$. This idea is known as the *Least Square approach*. This approach was used for traditional – shallow – neural networks. However, empirically, it turns out that for deep learning, it is more effective to use Kullback-Leibler (KL) divergence

$$-\sum_{k=1}^{K}[y_k \cdot \log(z_k) + (1 - y_k) \cdot \log(1 - z_k)].$$

In this formula, the values y_k and z_k are normalized, so that they are all located in the interval $[0, 1]$. This formula makes sense when y_k and z_k are probabilities, but in most applications of machine learning, they are not probabilities – so why is this minimized expression so effective?

A decision-related explanation for this effectiveness is provided in [35]. The ultimate goal of all computations is to make a decision. From this viewpoint, ideally, we should always have values z_k equal to 0 or 1, corresponding to two possible decisions. In practice, a neural network – or any other machine learning tool – produces a value which is different from 0 and 1. Usually, to make a decision, we select a threshold r: if $z_k > r$, we make a decision corresponding to 1, otherwise we make a decision corresponding to 0.

In different situations, we may use different thresholds. In many practical situations, it makes sense to select $r = 0.5$. However, in many other situations, it makes sense to select r close to 1 – e.g., if we are deciding on whether to undergo a potentially dangerous surgical operations, it makes sense to only do it when we are almost absolutely sure that this operation is needed. On the

other hand, if there is a even a small risk of a hacker attack on an important computer system, then it makes sense to invoke all our defenses – in this case, the threshold is close to 0.

Since we do not have an a priori reason to believe that some thresholds are more reasonable – and thus, more frequent – it makes sense to assume that all threshold values from the interval $[0, 1]$ are equally frequent, i.e., that the value r is uniformly distributed on the interval $[0, 1]$. This means that in general, if we encounter similar (N) situation with the resulting value z in different setting, with different thresholds r, then with probability z we will make a positive decision and with probability $1 - z$ a negative decision. In other words, we will have $N \cdot z$ positive decisions and $N \cdot (1 - z)$ negative decisions. The actual probability that the decision is correct is determined by the actual (unknown) value y which is, in general, somewhat different from z. We can us simple probability formulas to compute the probability that the z-based arrangement will only contain correct decisions. It turns out that maximizing this probability is equivalent to maximizing the KL divergence.

10.7 Selecting initial weights

Empirically, the best idea is to use initial weights close to equal ones; this empirical fact is explained in [1].

10.8 How to optimize the selected objective function: why gradient descent?

In neural networks, the main way to find the optimum of the objective function turns out to be gradient descent – namely, a specific algorithm implementation of gradient descent which is known as *backpropagation*. This is somewhat puzzling, since the experience of numerical optimization is that other methods are much more effective. This puzzle is explained in [15].

Indeed, gradient descent works well when we are reasonably far away from the actual minimum – and therefore, the gradient is different from 0. The problem with the gradient descent is that when we get close to the minimum – i.e., to the point when the gradient is 0 – the gradient becomes very small. As a result, gradient descent – where the changes are proportional to the gradient – becomes very slow. Thus, in numerical methods, optimization methods that take second derivatives into account are much faster.

A specific feature of machine learning is that, in contrast to numerical methods, when we *do* need to find the exact minimum, in machine learning, we *do not* want the exact minimum. The exact minimum would mean that we fit all the training data perfectly. However, these data contain measurement errors, so if we fit the data perfectly, we *overfit*, we fit the noise. For example, if we have an approximately linear dependence and we have 3 observations, we can fit them perfectly well with a quadratic curve – but it will be very wrong for large inputs. In statistics, we do not want the perfect fit, we want, e.g., the sum of the squares of all the deviations – from the ideal fit – to reach its most probably

value – which is number of observations times the variance of the measurement error.

Since we do not want to go near the actual minimum, gradient descent will work well.

10.9 Why all this works?

Challenge. Empirically, it all works well. However, if we want to understand why it all works, we face a challenge.

In general, the more unknowns in a problem, the more computations we need to find these unknowns. However, in deep learning, the opposite effect has been observed: learning becomes more efficient when the number of parameters is much larger than the number of equations – i.e., then the number of data points that we try to match.

Our explanations. We have three related explanations for this challenge.

Two qualitative explanations for this paradoxical situations are presented in [18]:

- The first explanation is that with the large number of parameters – neural weights – whose initial values are random, the objective function becomes random. It can be proven that a random function almost always attains its optimum at a single point – and it is known that, in general, optimization problems with this uniqueness property are easier to solve.
- The second explanation is related to the fact that training a neural network is done by back-propagation – which, from the mathematical viewpoint, is simply a gradient descent. The larger the gradient, the faster the minimized objective function decreases – and the more unknowns, the larger the size of the gradient.

The third explanation is provided in [76]. This explanation is more mathematical, it is based on the fact that neural networks are non-linear. To explain how non-linearity explains this challenge, this paper uses a mathematical result known as Chevalley-Warning Theorem – a result that was motivated by Fermat Last Theorem.

10.10 How to best combine results of several trainings

To speed up training, deep learning algorithms use the idea of a so-called *dropout* – they train several sub-networks on different parts of the data, and then "average" the results $r_1, \ldots, r_m$. Empirically, geometric average $\sqrt[m]{r_1 \cdot \ldots \cdot r_m}$ works the best. In [29, 39, 45], this empirical success is explained by scaling.

10.11 How to make decisions based on the results of the neural network: why softmax?

In neural networks or classification, we usually have, in effect, a special neural network for each class i that estimates the strength s_i of evidence that a given object belongs to this class. These values are then transformed into probabilities p_i of belonging to different classes. Empirically the most effective transformation is *softmax*:

$$p_i = \frac{\exp(a \cdot s_i)}{\sum\limits_j \exp(a \cdot s_j)}.$$

Its empirical success can be explained by invariance which is similar to scale-invariance: invariance with respect to selecting the starting point for measurements [39, 45].

10.12 Can all this be practically useful?

Most of the results that we cited so far simply provide theoretical justification for features and techniques that have already been empirically proven to be successful. Yes, the presence of such theoretical explanations makes us more confident in using these techniques. However, a natural question is: can it help to come up with better techniques?

This question was often raised by a Russian Nobel-prize winning physicist Lev Landau who dismissed foundational papers with no practical application by calling them *Neue Begrundung* – New Foundations – which were the typical first words of such a paper's title.

In our defense, we can mention that in physics, symmetry methods have indeed led to interesting theories and interesting discoveries – to the extent that, in contrast to Newton's time when new theories were formulated in terms of differential equations, now many theories are described in terms of their symmetries and invariances – and differential equations follow from these invariances; see, e.g., [26, 72]. One of the first such symmetry-motivated theories was the theory of quarks – see our related comment in the next section.

We cannot yet brag of similar *major* successes in neural networks, but we can brag about *some* successes in this direction: e.g., in [3], we showed that symmetry-motivated pre-processing can help a neural network to better diagnose different lung disfunctions in children.

11 Applications beyond machine learning

11.1 What we do in this section

Similar ideas and results can be used to explain empirical success of other data processing techniques – techniques not necessarily related to machine learning – both in general and in applications to specific application areas.

We start this section with techniques related to uncertainty processing. Data for processing comes either from measurements or from experts – mostly from measurements. So, we start this section with dealing with uncertainty of measurement results – which, as we will explain, often results in interval uncertainty. Then, we deal with uncertainty of expert information – techniques for dealing with this information are known as *fuzzy* techniques.

After that, we provide two examples of other data processing techniques. Finally, we provide two examples of specific applications: to the study of the physical world and to decision making.

11.2 Applications to interval techniques

What is the problem. Computers process data. Most data either comes from measurements – or from the previously performed processing of measurement results. Measurements are never absolutely accurate, there is usually some difference $\Delta x \stackrel{\text{def}}{=} \widetilde{x} - x$ between the measurement result $\widetilde{x}$ and the actual (unknown) value x of the corresponding physical quantity. The manufacturer of the measuring instrument usually provides an upper bound Δ on the absolutely value of the measurement error. Indeed, if there was no such bound, this means that no matter what the measurement result is, the actual value can be as large as theoretically possible – and this will be not a measurement, but a wild guess.

Ideally, we should also know how frequent are different values of the measurement error from the interval $[-\Delta, \Delta]$. However, determining the corresponding probabilities requires a lot of testing, and such testing is rarely done for run-of-the mill sensors. So, in many practical situations, all we know about the measurement error of each measurement $\widetilde{x}_i$ is the corresponding upper bound Δ_i.

We process this data, i.e., we apply some algorithm $f(x_1, \ldots, x_n)$ to the measurement results $\widetilde{x}_i$ and get the result $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$. Since the measurement results are somewhat different from the actual values, the result $\widetilde{y}$ of processing measurement results is, in general, somewhat different from the result $y = f(x_1, \ldots, x_n)$ that we would have gotten if we know the exact values x_i of the input quantities.

In many practical applications, it is important to know how accurate is the estimate $\widetilde{y}$, i.e., how big can be the difference $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y$. For example, when an oil company wants to decide whether to invest into a prospective oil field, it estimates the amount of oil. If this estimate is $\widetilde{y} = 150$ million tons and Δy does not exceed 10 million, then we should start drilling. However, if Δy can be as large as 200 million, then maybe there is no oil at all – so more estimates are needed.

In general, this problem is NP-hard, but for many practical cases, there are feasible algorithms. In general, the problem of estimating the set of possible values of $y = f(x_1, \ldots, x_n)$ when about each x_i we only know it belongs to the interval $[\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$ is known as the problem of *interval computations*; see, e.g., [32, 55, 59, 61]. In general, this problem is NP-hard; see, e.g., [49].

However, in practical situations, the measurement errors are small. Then, we can safely ignore the terms which are quadratic (or of higher order) in terms of Δx_i and keep only linear terms. For example, even for a non-very-accurate measurement, with accuracy of 10%, the square of this term is 1%, which is much smaller than 10%.

If we only keep linear terms, then we get

$$\Delta y = \widetilde{y} - y = f(\widetilde{x}_1, \ldots, \widetilde{x}_n) - f(\widetilde{x}_1 - \Delta x_1, \ldots, \widetilde{x}_n - \Delta x_n) \approx \sum_{i=1}^{n} c_i \cdot \Delta x_i,$$

where

$$c_i \stackrel{\text{def}}{=} \frac{\partial f}{\partial x_i}_{x_i = \widetilde{x}_i}.$$

One can check that when Δx_i take values from the interval $[-\Delta_i, \Delta_i]$, then Δy takes values from the interval $[-\Delta, \Delta]$, where

$$\Delta \stackrel{\text{def}}{=} \sum_{i=1}^{n} |c_i| \cdot \Delta_i.$$

Additional complication: need to deal with black-box algorithms. When we have an explicit expression for the data processing algorithm, we can explicitly find all the derivatives c_i. However, in many practical situations, we use commercial packages for data processing, for which the data processing algorithm is proprietary – for us, it is a black box. In these case, we cannot compute the derivatives analytically, we can only find them numerically, as

$$c_i \approx \frac{f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n) - \widetilde{y}}{h},$$

for some small h. However, this requires $n+1$ calls to the algorithm f: one call to compute Δy, and n calls to compute the values $f(\ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h, \widetilde{x}_{i+1}, \ldots)$. In situations like oil prospecting, we process thousands of data points, and each call to f may take hours on a high performance computer. In such situations, repeating such calls thousands of times is not realistic.

Cauchy deviates: a mathematical trick that needs a commonsense explanation. There is a mathematical trick that helps speed up computations; see, e.g., [43]. It is based on the fact that if each input Δx_i is Cauchy distributed with parameter Δ_i, then the resulting values Δy are also Cauchy distributed, with exactly the desired parameter Δ. This is a trick since Cauchy distribution with parameter Δ_i is not limited to the interval $[-\Delta_i, \Delta_i]$. It is therefore desirable to have some intuitive understanding of the resulting Cauchy deviate method.

Neural network ideas can lead to such an explanation. It turns out that neural network ideas can lead to such an explanation [50].

Relation to interval fields. The ideas explaining why we need neural networks in the first place can also explain the empirical success of *interval field* technique [10] – one of the techniques for dealing with interval-valued fields. In this technique, we fix several basic fields, and approximate a general field by a linear combination of the basic fields, with interval-valued coefficients. The same ideas were used in [10] to provide possible ways to further improve this technique.

11.3 Applications to fuzzy techniques

Why some versions of fuzzy techniques are more empirically successful. In fuzzy techniques (see, e.g., [12, 34, 60, 64, 65, 78]), many empirically successful selections of membership functions, "and"- and "or"-operations, operations corresponding to hedges, and defuzzification procedures can be explained by the desired to have the fastest possible computations [54, 77].

In [37], it is shown that two more empirically successful formulas can be explained by the desire to have the fastest possible computations:

- a fuzzy version of the "k out of n condition" – typical in medical diagnostics, and
- Zadeh's extension of the notion of set cardinality to the case of fuzzy sets.

Scaling – in particular, non-linear scaling – also explains the empirical fact that the best way of using fuzzy degrees in training a neural network is to first apply a sigmoid transformation $x \mapsto 1/(1+\exp(-x))$ to these degrees [5].

Why some algorithms for processing fuzzy techniques are empirically successful. Processing of fuzzy data is usually reduced to processing so-called alpha-cuts: intervals formed by all the values x for which the degree of confidence is larger than or equal to some value α. Since, as we have mentioned, neural network ideas can help explain techniques for processing intervals, it can therefore explain why these techniques are used in the fuzzy case as well.

When traditional fuzzy techniques work better and when more complex (e.g., type-2) techniques work better? In the traditional fuzzy techniques, experts' degrees of confidence in different imprecise statements like "x is small" are described by numbers. However, similarly to the fact that the expert cannot produce the exact estimate, he/she can only say that x is small, the same expert cannot naturally provide a single number describing his/her degree of confidence: it is 0.80 or 0.81? A more natural idea is to ask the expert to provide an *interval* of possible degree – or even an imprecise ("fuzzy") statement describing his/her degree. This idea is known as *type-2 fuzzy sets*.

Type-2 fuzzy sets provide a more adequate description of the expert's opinion. However, since now we need at least two numbers to describe each expert's degree – e.g., endpoints of the expert-provided interval – processing type-2 values takes more time. Empirically, sometimes, the speed is the main problem, so type-1 methods work better; in other cases, time is not that restricted, so type-2

methods are better. It is desirable to have general recommendations on when to use faster type-1 techniques and when to use slower-but-more-accurate type-2 fuzzy techniques. In [47], neural-related ideas are used to provide an answer to this question.

11.4 Applications to other data processing techniques: examples related to statistical and heuristic techniques

Application to statistical techniques. Arguments similar to the ones that we used to explain why neural networks are effective can be used to explain the empirical success of so-called "surrogate' statistical techniques [68].

Similar arguments can also explain the empirical success of Karhunen-Loéve (KL) decomposition, when an unknown function is represented as a linear combination to eigenfunctions of the covariance – and suggest ways to further improve this technique [10].

Application to heuristic techniques. In many practical situations, linear approximation is not very accurate, but, e.g., quadratic approximations require too many parameters and are, thus, too time-consuming. In such situations, one of the empirically successful techniques are Ordered Weighted Average (OWA) aggregation operation that transform values $x_1, \ldots, x_n$ into

$$w_1 \cdot x_{(1)} + \ldots + w_n \cdot x_{(n)},$$

where $x_{(i)}$ are the ordering of the inputs x_i: $x_{(1)} \le \ldots \le x_{(n)}$.

Invariance ideas explain the empirical success of these techniques [42].

11.5 Applications to physics

As we have mentioned earlier, neural networks have a universal *approximation* property: any continuous function on a bounded domain can be approximated, with any desired accuracy, by a neural network. It is also true that we have a stronger universal *representation* property – according to Kolmogorov theorem, any continuous function on a bounded domain can be exactly represented as a composition of addition and functions of one variable – which is, as we have mentioned, exactly neural networks.

As shown in [40], this result enables us to describe any interaction between particles in terms of pairwise interaction between the original particle and fictitious "particles" – just like a 3-chemicals reaction $a + b + c \to d$ can be formally represented as a sequence of two pairwise reactions $a + b \to ab$ and $ab + c \to d$, with a "fictitious" substance ab. It turns out that for this representation, we need to add 3 new "fictitious" particles for each newly added real particle. This seems in good accordance with the fact that, according to modern physics [26, 72]:

- protons and neutrons – some of the basic particles of the Universe that carry most mass of the normal matter – consist of 3 sub-particles (quarks) each, and

- these quarks are, in some sense, "fictitious" particles: while we can separate electrons, protons, and neutrons from an atom, we cannot separate quarks from each other; this property is known as *quark confinement*.

11.6 Applications to group decision making

Let us describe group decision making in precise terms. The decision of a group depends on the preferences of all the participants. According to decision theory (see, e.g., [27, 28, 41, 58, 62, 63, 66]), preferences of a rational person i ($i = 1, 2, \ldots, m$) can be described by assigning, to each possible alternative j ($j = 1, 2, \ldots, n$) a numerical value $u_{i,j}$ called its *utility*. Utility is usually selected in such a way that the i-th person's utility u_i of a situation in which we get each alternative j with probability p_j is equal to $u_i = p_1 \cdot u_{i,1} + \ldots + p_n \cdot u_{i,n}$.

In practice, in most decision making situations – e.g., dividing the inheritance – when it is not possible to divide absolutely equally, there is always possibility to reach equality if other participants pay some money to the participant who gets a smaller portion. Utility is usually non-linearly related to money, so that the money equivalent to the utility u_i is equal to $M_i(u_i)$ for some non-linear function $M_i(u_i)$. In this arrangement, the best alternative $p = (p_1, \ldots, p_n)$ is the one for which the overall equivalent money value $M(p_1, \ldots, p_n)$ – described by the sum of money values – is the largest:

$$M(p_1, \ldots, p_n) = \sum_{i=1}^{m} M_i(p_1 \cdot u_{i,1} + \ldots + p_n \cdot u_{i,n}).$$

How this is related to a neural network. The paper [73] noticed that this expression is exactly the general expression for a function computed by a shallow neural network – i.e., a neural network with a single hidden layer. According to [73], this fact has both negative and positive consequences.

Negative consequence. Since shallow neural networks are universal approximators, this means that the resulting function can be any continuous function – and thus, it can be as complex as possible, we cannot expect a feasible algorithm for solving all particular cases of this problem.

Positive consequence. A positive consequence is that since the corresponding function is the same as for neural networks, then, to find a solution, we can use techniques that work well for neural networks – such as back-propagation.

12 Instead of conclusion

In this paper, we provided an explanation for many empirically successful neural techniques – and cited many papers where other empirical successes are similarly explained. There are so many things explained that at first glance, it may sound as if most empirical successes have already been explained – especially if one takes into account that we focused mostly on explanations in which we

ourselves participated, and other researchers have produced many alternative explanations for these and other empirical phenomena.

However, the above optimistic impression is false. Only a small proportion of empirical successes has been explained – and many of these explanations are still more on the qualitative level, they need to be made quantitative.

Also, deep learning applications – in particular, applications to decision making – are a booming field. Many new empirical successes are reported every week, maybe even every day – and, of course, it will be helpful if we can explain them. More explanations are needed, and we hope that this paper will inspire researchers to come up with such explanations!

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).

The authors are very thankful to the editors of this book for their kind invitation and for their valuable suggestions.

References

[1] D. Aguirre, P. Hassoun, R. Lopez, C. Serrano, M. R. Soto, A. Torres, and V. Kreinovich, "Smaller standard deviation for initial weights improves performance of classifying neural networks: a theoretical explanation of unexpected simulation results", *Journal of Uncertain Systems*, 2019, Vol. 13, No. 3, pp. 168–171.

[2] K. Alvarez, J. C. Urenda, O. Csiszár, G. Csiszár, J. Dombi, G. Eigner, and V. Kreinovich, "Towards fast and understandable computations: which 'and'- and 'or'-operations can be represented by the fastest (i.e., 1-layer) neural networks? which activations functions allow such representations?", *Acta Polytechnica Hungarica*, 2021, Vol. 18, No. 2, pp. 27–45.

[3] N. Avila, J. Urenda, N. Gordillo, V. Kreinovich, and S. Shahbazova, "Scale-invariance-based pre-processing drastically improves neural network learning: case study of diagnosing lung dysfunction in children", In: S. N. Shahbazova, A. M. Abbasov, V. Kreinovich, J. Kacprzyk, and I. Batyrshin (eds.), *Recent Developments and the New Directions of Research, Foundations, and Applications*, Springer, Cham, Switzerland, 2023, Vol. 2, pp. 171–192.

[4] C. Baral, O. Fuentes, and V. 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.

[5] C. Baral and V. Kreinovich, "Why sigmoid transformation helps incorporate logic into deep learning: a theoretical explanation", In: M. Ceberio and V. Kreinovich (eds.), *Uncertainty, Constraints, AI, and Decision Making*, Springer, Cham, Switzerland, to appear.

[6] B. Bede, O. Kosheleva, and V. Kreinovich, "Every ReLU-based neural network can be described by a system of Takagi-Sugeno fuzzy rules: a theorem", In: H. T. Nguyen, J. Kacprzyk, and V. Kreinovich (eds.), *Contributions of Fuzzy Techniques to Systems and Control: A Tribute to Michio Sugeno*, Springer, Cham, Switzerland, to appear.

[7] B. Bede, V. Kreinovich, and U. Pham, "Why Rectified Linear Unit is efficient in machine learning: one more explanation", In: V. Kreinovich, S. Sriboonchitta, and W. Yamaka (eds.), *Machine Learning for Econometrics and Related Topics*, Springer, Cham, Switzerland, 2024, pp. 161-167.

[8] B. Bede, V. Kreinovich, and P. Toth, "Equivalence between 1-D Takagi-Sugeno fuzzy systems with triangular membership functions and neural networks with ReLU activation", In: K. Cohen, N. Ernest, B. Bede, and V. Kreinovich (Eds.), *Fuzzy Information Processing 2023, Proceedings of the 2023 Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'2023*, Cincinnati, Ohio, May 31 – June 2, 2023, Springer Lecture Notes in Networks and Systems, 2023, Vol. 751, pp. 44–56.

[9] B. Bede, V. Kreinovich, and P. Toth, "Equivalence between TSK fuzzy systems with triangular membership functions and neural networks with ReLU activation on the real line", *Proceedings of the NAFIPS International Conference on Fuzzy Systems, Soft Computing, and Explainable AI NAFIPS'2024*, South Padre Island, Texas, May 27–29, 2024.

[10] M. Beer, O. Kosheleva, and V/ Kreinovich, "Uncertainty: ideas behind neural networks lead us beyond KL-decomposition and interval fields", *Proceedings of the IEEE Series of Symposia on Computational Intelligence SSCI'2021*, Orlando, Florida, December 4–7, 2021.

[11] M. Beer, J. Urenda, O. Kosheleva, and V. Kreinovich, "Why spiking neural networks are efficient: a theorem", In: M.-J. Lesot, S. Vieira, M. Z. Reformat, J. P. Carvalho, A. Wilbik, B. Bouchon-Meunier, and R. R. Yager (eds.), *Proceedings of the 18th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems IPMU'2020*, Lisbon, Portugal, June 15–19, 2020, pp. 59–69.

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

[13] L. Bokati, O. Kosheleva, V. Kreinovich, and A. Sosa, "Why deep learning is more efficient than Support Vector Machines, and how it is related to sparsity techniques in signal processing", *Proceedings of the 2020 4th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence ISMSI'2020*, Thimpu, Bhutan, April 18–19, 2020.

[14] L. Bokati, V. Kreinovich, J. Baca, and N. Rovelli, "Why Rectified Power (RePU) activation functions are efficient in deep learning: a theoretical explanation", In: M. Ceberio and V. Kreinovich (eds.), *Uncertainty, Constraints, and Decision Making*, Springer, Cham, Switzerland, 2023, pp. 7–13.

[15] J. Contreras, M. Ceberio, O. Kosheleva, and V. Kreinovich, "why gradient descent – not the best optimization technique – works best in neural networks: qualitative explanation", *Journal of Combinatorics, Information, and System Sciences JCISS*, 2020, Vol. 45, No. 1–4, pp. 1–10.

[16] 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.

[17] J. Contreras, M. Ceberio, O. Kosheleva, V. Kreinovich, and N. H. Phuong, "Why Rectified Linear neurons: two convexity-related explanations", In: N. H. Phuong and V. Kreinovich (eds.), *Biomedical and Other Applications of Soft Computing*, Springer, Cham, Switzerland, 2023, pp. 41–47.

[18] J. Contreras, M. Ceberio, O. Kosheleva, V. Kreinovich, and N. H. Phuong, "Computational paradox of deep learning: a qualitative explanation", In: N. H. Phuong and V. Kreinovich (eds.), *Deep Learning and Other Soft Computing Techniques: Biomedical and Related Applications*, Springer, Cham, Switzerland, 2023, pp. 245–252.

[19] J. Contreras, M. Ceberio, and V. Kreinovich, "Why dilated Convolutional Neural Networks: a proof of their optimality", *Entropy*, 2021, Vol. 23, Paper 767.

[20] J. Contreras, M. Ceberio, and V. Kreinovich, "One more physics-based explanation for rectified linear neurons", In: M. Ceberio and V. Kreinovich (eds.), *Uncertainty, Constraints, and Decision Making*, Springer, Cham, Switzerland, 2023, pp. 195–198.

[21] J. Contreras, M. Ceberio, and V. Kreinovich, "Why rectified linear neurons: a possible interval-based explanation", In: N. N. Thach, N. D. Trung, D. T. Ha, and V. Kreinovich (eds.), *Artificial Intelligence and Machine Learning for Econometrics: Applications and Regulation (and Related Topics)*, Springer, Cham, Switzerland, to appear.

[22] D. Cruz, R. Godoy, and V. Kreinovich, "Why, in deep learning, non-smooth activation function works better than smooth ones", In: M. Ceberio and

V. Kreinovich (eds.), *Decision Making under Uncertainty and Constraints: A Why-Book*, Springer, Cham, Switzerland, 2023, pp. 111–115.

[23] O. Csiszár, L. S. Pusztaházi, L. Dénes-Fazakas, M. S. Gashler, V. Kreinovich, and G. Csiszár, "Uninorm-like parametric activation functions for human-understandable neural models", *Knowledge-Based Systems*, 2023, Vol. 260, Paper 110095, https://doi.org/10.1016/j.knosys.2022.110095

[24] L. Denes-Fazakas, L. Szilagyi, G. Eigner, O. Kosheleva, M. Ceberio, and V. Kreinovich, "Which activation function works best for training artificial pancreas: empirical fact and its theoretical explanation", *Proceedings of the IEEE Series of Symposia on Computational Intelligence SSCI 2023*, Mexico City, Mexico, December 6–8, 2023.

[25] A. Farhan, O. Kosheleva, and V. Kreinovich, "Why max and average poolings are optimal in convolutional neural networks", In: H. Seki, C. H. Nguyen, V.-N. Huynh, and M. Inuiguchi (eds.), *USB Proceedings of the 7th International Symposium on Integrated Uncertainty in Knowledge Modelling and Decision Making IUKM'2019*, Nara, Japan, March 27–29, 2019, pp. 1–10.

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

[27] P. C. Fishburn, *Utility Theory for Decision Making*, John Wiley & Sons Inc., New York, 1969.

[28] P. C. Fishburn, *Nonlinear Preference and Utility Theory*, The John Hopkins Press, Baltimore, Maryland, 1988.

[29] A. Gholamy, J. Parra, V. Kreinovich, O. Fuentes, and E. Anthony, "How to best apply deep neural networks in geosciences: towards optimal "averaging" in dropout training", In: J. Watada, S. C. Tan, P. Vasant, E. Padmanabhan, and L. C. Jain (eds.), *Smart Unconventional Modelling, Simulation and Optimization for Geosciences and Petroleum Engineering*, Springer Verlag, 2019, pp. 15–26.

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

[31] S. Holguin and V. Kreinovich, "Why residual neural networks", In: M. Ceberio and V. Kreinovich (eds.), *Decision Making under Uncertainty and Constraints: A Why-Book*, Springer, Cham, Switzerland, 2023, pp. 117–120.

[32] L. Jaulin, M. Kiefer, O. Didrit, and E. Walter, *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control, and Robotics*, Springer, London, 2012.

[33] R. B. Kearfott and V. Kreinovich, "Beyond convex? global optimization is feasible only for convex objective functions: a theorem", *Journal of Global Optimization*, 2005, Vol. 33, No. 4, pp. 617–624.

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

[35] O. Kosheleva and V. Kreinovich, "Why deep learning methods use KL divergence instead of least squares: a possible pedagogical explanation", *Mathematical Structures and Modeling*, 2018, Vol. 46, pp. 102–106.

[36] O. Kosheleva and V. Kreinovich, "Symmetry approach explains why some choices in computational intelligence work better", *Abstracts of the Conference on Soft Computing and Intelligent Systems with Applications*, Gyor, Hungary, June 5-7, 2025.

[37] O. Kosheleva, V. Kreinovich, and H. P. Nguyen, "How to describe conditions like 2-out-of-5 in fuzzy logic: a neural approach", *Journal of Advanced Computational Intelligence and Intelligent Informatics (JACIII)*, 2020, Vol. 24, No. 5, pp. 593–598.

[38] Vladik Kreinovich, "S. Maslov's iterative method: 15 years later (freedom of choice, neural networks, numerical optimization, uncertainty reasoning, and chemical computing)", In: V. Kreinovich and G. Mints (eds.), *Problems of reducing the exhaustive search*, American Mathematical Society, Providence, RI, 1997, pp. 175–189.

[39] 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*, Springer, Cham, Switzerland, 2021, pp. 387–397.

[40] V. Kreinovich, "Fundamental properties of pair-wise interactions naturally lead to quarks and quark confinement: a theorem motivated by neural universal approximation results", In: M. Ceberio and V. Kreinovich (eds.), *How Uncertainty-Related Ideas Can Provide Theoretical Explanation for Empirical Dependencies*, Springer, Cham, Switzerland, 2021, pp. 75–81.

[41] V. Kreinovich, "Decision making under interval uncertainty (and beyond)", In: P. Guo and W. Pedrycz (eds.), *Human-Centric Decision-Making Models for Social Sciences*, Springer Verlag, 2014, pp. 163–193.

[42] V. Kreinovich, "Ordered Weighted Averaging (OWA), decision making under uncertainty, and deep learning: how is this all related?", *Information*, 2022, Vol. 13, No. 2, Paper 82.

[43] V. Kreinovich and S. Ferson, "A new Cauchy-based black-box technique for uncertainty in risk analysis", *Reliability Engineering and Systems Safety*, 2004, Vol. 85, No. 1–3, pp. 267–279.

[44] V. Kreinovich and O. Fuentes, "High-concentration chemical computing techniques for solving hard-to-solve problems, and their relation to numerical optimization, neural computing, reasoning under uncertainty, and freedom of choice", In: E. Katz (ed.), *Molecular and Supramolecular Information Processing: From Molecular Switches to Logical Systems*, Wiley-VCH, Wienheim, Germany, 2012, pp. 209–235.

[45] 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*, Springer, Cham, Switzerland, 2021, pp. 195–220.

[46] V. Kreinovich and O. Kosheleva, "Deep learning (partly) demystified", *Proceedings of the 2020 4th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence ISMSI'2020*, Thimpu, Bhutan, April 18–19, 2020.

[47] V. Kreinovich and O. Kosheleva, "Fuzzy or neural, type-1 or type-2 – when each is better: first-approximation analysis", In: Y. P. Kondratenko, V. Kreinovich, W. Pedrycz, A. A. Chikrii, and A. M. Gil Lafuente (eds.), *Artificial Intelligence in Control and Decision-Making Systems*, Springer, 2023, pp. 67–74.

[48] V. Kreinovich, O. Kosheleva, and A. Ortiz-Muñoz, "Need for simplicity and everything is a matter of degree: how Zadeh's philosophy is related to Kolmogorov complexity, quantum physics, and deep learning", In: S. N. Shahbazova, A. M. Abbasov, V. Kreinovich, J. Kacprzyk, and I. Batyrshin (eds.), *Recent Developments and the New Directions of Research, Foundations, and Applications*, Springer, Cham, Switzerland, 2023, Vol. 1, pp. 203–215.

[49] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1998.

[50] V. Kreinovich and H. T. Nguyen, "Towards a neural-based understanding of the Cauchy deviate method for processing interval and fuzzy uncertainty", *Proceedings of the 2009 World Congress of the International Fuzzy Systems Association IFSA*, Lisbon, Portugal, July 20–24, 2009, pp. 1264–1269.

[51] V. Kreinovich, H. T. Nguyen, and S. Sriboonchitta, "Need for data processing naturally leads to fuzzy logic (and neural networks): fuzzy beyond experts and beyond probabilities", *International Journal of Intelligent Systems*, 2016, Vol. 31, No. 3, pp. 276–293.

[52] V. Kreinovich and C. Quintana. "Neural networks: what non-linearity to choose?," *Proceedings of the 4th University of New Brunswick Artificial Intelligence Workshop*, Fredericton, N.B., Canada, 1991, pp. 627–637.

[53] V. Kreinovich, O. Sirisaengtaksin, and S. Cabrera, "Wavelet neural networks are optimal approximators for functions of one variable." *Proceedings of the IEEE International Conference on Neural Networks*, Orlando, Florida, July 1994, Vol. 1, pp. 299-303.

[54] V. Kreinovich and D. Tolbert, "Minimizing computational complexity as a criterion for choosing fuzzy rules and neural activation functions in intelligent control". In: M. Jamshidi, C. Nguyen, R. Lumia, and J. Yuh (eds.), *Intelligent Automation and Soft Computing. Trends in Research, Development, and Applications. Proceedings of the First World Automation Congress (WAC'94), August 14–17, 1994, Maui, Hawaii*, TSI Press, Albuquerque, NM, 1994, Vol. 1, pp. 545–550.

[55] B. J. Kubica, *Interval Methods for Solving Nonlinear Constraint Satisfaction, Optimization, and Similar Problems: from Inequalities Systems to Game Solutions*, Springer, Cham, Switzerland, 2019.

[56] M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*, Springer, Cham, Switzerland, 2019.

[57] R. Lozano, I. Montoya Sanchez, and V. Kreinovich, "Why deep neural networks: yet another explanation", In: M. Ceberio and V. Kreinovich (eds.), *Uncertainty, Constraints, and Decision Making*, Springer, Cham, Switzerland, 2023, pp. 199–202.

[58] R. D. Luce and R. Raiffa, *Games and Decisions: Introduction and Critical Survey*, Dover, New York, 1989.

[59] G. Mayer, *Interval Analysis and Automatic Result Verification*, de Gruyter, Berlin, 2017.

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

[61] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*, SIAM, Philadelphia, 2009.

[62] H. T. Nguyen, O. Kosheleva, and V. Kreinovich, "Decision making beyond Arrow's 'impossibility theorem', with the analysis of effects of collusion and mutual attraction", *International Journal of Intelligent Systems*, 2009, Vol. 24, No. 1, pp. 27–47.

[63] H. T. Nguyen, V. Kreinovich, B. Wu, and G. Xiang, *Computing Statistics under Interval and Fuzzy Uncertainty*, Springer Verlag, Berlin, Heidelberg, 2012.

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

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

[66] H. Raiffa, *Decision Analysis*, McGraw-Hill, Columbus, Ohio, 1997.

[67] S. Robles Herrera, M. Ceberio, and V. Kreinovich, "When is deep learning better and when is shallow learning better: qualitative analysis", *International Journal of Parallel, Emergent and Distributed Systems*, 2022, DOI: 10.1080/17445760.2022.2070748.

[68] S. Robles Herrera, M. Ceberio, and V. Kreinovich, "Foundations of neural networks explain the empirical success of the 'surrogate' approach to ordinal regression – and recommend what next", In: M. Ceberio and V. Kreinovich (eds.), *Uncertainty, Constraints, AI, and Decision Making*, Springer, Cham, Switzerland, to appear.

[69] E. D. Rodriguez Velasquez, O. Kosheleva, and V. Kreinovich, "Invariance-based approach explains empirical formulas from pavement engineering to deep learning", *Advances in AI and Machine Learning*, 2022, Vol. 2, No. 3, pp. 456–468.

[70] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, Chapman and Hall/CRC, Boca Raton, Florida, 2011.

[71] O. Sirisaengtaksin, V. Kreinovich, and H. T. Nguyen, "Sigmoid neurons are the safest against additive errors", *Proceedings of the First International Conference on Neural, Parallel, and Scientific Computations*, Atlanta, Georgia, May 28–31, 1995, Vol. 1, pp. 419–423.

[72] K. S. Thorne and R. D. Blandford, *Modern Classical Physics: Optics, Fluids, Plasmas, Elasticity, Relativity, and Statistical Physics*, Princeton University Press, Princeton, New Jersey, 2021.

[73] R. Trejo and V. Kreinovich, "Complexity of collective decision making explained by neural network universal approximation theorem", In: G. Alefeld and R. A. Trejo (eds.), *Interval Computations and its Applications to Reasoning Under Uncertainty, Knowledge Representation, and Control Theory. Proceedings of MEXICON'98, Workshop on Interval Computations, 4th World Congress on Expert Systems*, México City, México, 1998.

[74] J. C. Urenda, O. Csiszar, G. Csiszar, J. Dombi, O. Kosheleva, V. Kreinovich, and G. Eigner, "Why squashing functions in multi-layer neural networks", *Proceedings of the 2020 IEEE International Conference on Systems, Man, and Cybernetics SMC'2020*, Toronto, Canada, October 11–14, 2020, pp. 1705–1711.

[75] J. C. Urenda, O. Kosheleva, and V. Kreinovich, "Fuzzy techniques explain the effectiveness of relu activation function in deep learning", In: P. Melin and O. Castillo (eds.), *Proceedings of the International Seminar on Computational Intelligence ISCI'2023*, Tijuana, Mexico, August 30–31, 2023.

[76] J. C. Urenda, O. Kosheleva, and V. Kreinovich, "Why deep learning is under-determined? why usual numerical methods for solving partial differential equations do not preserve energy? the answers may be related to Chevalley-Warning Theorem (and thus to Fermat Last Theorem)", In: M. Ceberio and V. Kreinovich (eds.), *Uncertainty, Constraints, AI, and Decision Making*, Springer, Cham, Switzerland, to appear.

[77] J. C. Urenda and V. Kreinovich, "Why Rectified Linear activation functions? Why max-pooling? A possible explanation", In: O. Castillo and P. Melin (eds.), *New Perspectives on Hybrid Intelligent System Design based on Fuzzy Logic, Neural Networks and Metaheuristics*, Springer, 2023, pp. 459–463.

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