5-2019

Optimization under Uncertainty Explains Empirical Success of Deep Learning Heuristics

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

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

Comments:
Technical Report: UTEP-CS-19-49

Optimization under Uncertainty Explains Empirical Success of Deep Learning Heuristics

Vladik Kreinovich and Olga Kosheleva
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
olgak@utep.edu, vladik@utep.edu

Abstract

One of the main objectives of science and engineering is to predict the future state of the world – and to come up with devices and strategies that would make this future state better. In some practical situations, we know how the state changes with time – e.g., in meteorology, we know the partial differential equations that describes the atmospheric processes. In such situations, prediction becomes a purely computational problem. In many other situations, however, we do not know the equation describing the system's dynamics. In such situations, we need to learn this dynamics from data. At present, the most efficient way of such learning is to use deep learning – training a neural network with a large number of layers. To make this idea truly efficient, several trial-and-error-based heuristics were discovered, such as the use of rectified linear neurons, softmax, etc. In this chapter, we show that the empirical success of many of these heuristics can be explained by optimization-under-uncertainty techniques.

1 Formulation of the Problem

Need for machine learning. One of the main objectives of science and engineering is:

- to predict the future state of the world, and
- to come up with devices and strategies that would make this future state better.

In some practical situations, we know how the state changes with time. For example, in meteorology, we know the partial differential equations that describes the atmospheric processes, and we can use these equations to predict tomorrow's weather. In such situations, prediction becomes largely a purely computational problem.

In many other situations, however, we do not know the equation describing the system's dynamics. In such situations, we need to learn this dynamics from data. The corresponding techniques are known as *machine learning*.

Neural networks: a brief reminder. Not only computers can learn, we humans can learn, and many animals can learn. So, to make machines learn, a reasonable idea is to emulate how we humans learn. All our mental activities – including learning – is performed by special interconnected cells called *neurons*. Thus, a reasonable idea is to have a network of devices simulating such neurons; such a network is known as an *artificial neural network*, or simply a *neural network*, for short.

Each biological neuron takes several input signals $x_1, \ldots, x_n$ and transforms them into an output. In the first approximation, signals are first linearly combined, with appropriate weights $w_0, w_1, \ldots, w_n$. Then, some nonlinear transformation $s_0(z)$ – known as the *activation function* – is applied to the resulting linear combination $w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n$: $y = s_0(w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n)$. (This activation function is usually non-decreasing.) In most neural network models, this is exactly how each neuron operates.

Signals from some neurons becomes inputs to other neurons, etc. First, neurons get output from the problem. The set of such neurons is usually called the *first* (*input*) layer. The outputs of these neurons are processed by other neurons – which form the *second layer*, etc., until we get the last (*output*) layer that will eventually generate the desired prediction.

Such neural networks have indeed turned out to be efficient machine learning tools; see, e.g., [4, 8, 16, 17].

Deep neural networks: main idea. In many cases, artificial neural networks are not yet as good as well-trained human experts. One of the main reasons for this is that while our brain uses billions of neurons, the computer systems do not yet have an ability to incorporate that many neurons. With the progress in computing hardware, however, we are able to incorporate more and more neurons – and thus, hopefully, get better and better performance.

In principle, we have several possible ways to add neurons to a network:

- we can place more neurons in each layer, or
- we can form new layers, or
- we can do both.

Researchers tried all three options, and found out that the best results are achieved if we add new layers – i.e., if we consider "deep" neural networks, with a large number of layers. This is the main idea behind *deep learning* and *deep neural networks*; see, e.g., [8].

This can easily be explained (see, e.g., [12]). Indeed, for the same number of variables, we want to get more accurate approximations. For given number of variables, and given accuracy, we get N possible combinations. If all combinations correspond to different functions, we can implement N functions.

However, if some combinations lead to the same function, we can implement fewer different functions.

If we have K neurons in a layer, then each of $K!$ permutations of these neurons retains the resulting function; see, e.g., [9]. Thus, instead of N functions, we only implement

$$\frac{N}{K!} \ll N \text{ functions.}$$

So, to increase approximation accuracy, we need to minimize the number K of neurons in each layer.

To get a good accuracy, we need many parameters, thus many neurons. Since each layer has few neurons, we thus need many layers.

Deep learning became a success. The main idea behind deep learning may be reasonable, but by itself, this idea does not necessarily lead to successful learning. Many improving ideas have been proposed, some worked, some did not. As a result of all this trial-and-error experimenting, at present, deep learning is the most efficient machine learning tool.

Let us briefly enumerate the main ideas that made deep learning a success.

First idea: what activation function $s_0(z)$ should we use? Originally, artificial neural networks used *sigmoid function* $s_0(z) = \dfrac{1}{1+\exp(-z)}$. The main reason for their use was that this is how most biological neurons work.

This activation function worked well for traditional neurons, but, somewhat surprisingly, for multi-layer deep network, it did not work at all. Several alternatives have been tried – until it turned out that the *rectified linear* activation function $s_0(z) = \max(0, z)$ works the best. Why is not fully clear. There is some understanding of why this new activation function is better than the original sigmoid function (see, e.g., [8]), but it is not clear why namely this activation function – and not many similar ones – works the best.

And, to add to the suspense, in some cases, it is efficient to add a layer of sigmoid neurons to the network consisting of rectified linear neurons. Why sigmoid and not anything else?

Second idea: need for pooling. If we use a neural network to process an image or a signal, we quickly realize that the corresponding input contains too many numerical values, and processing all the them makes computations last forever. For example, an average-quality image has 1,000,000 pixels – so we need 1,000,000 real numbers to describe this image. This requires too much computation time.

To decrease the size of the input and thus, make computations feasible, it is desirable to "pool" several inputs – e.g., inputs corresponding to nearby locations – into a single input value. Such "pooling" is ubiquitous in signal processing – e.g., to get a more accurate measurement result, we often measure the same quantity several times and take an average. Based on this experience, one would expect that for neural networks as well, the average pooling should work the best. Surprisingly, it turns out that in most situations, maximum pooling works much better. Why?

Third and fourth ideas: softmax instead of maximum and geometric mean instead of the usual average. At several stages during the learning process, we may get somewhat different results. For example, we can get different results at different stages of the training, or – if we want to speed up the learning process and run several stages in parallel – different results produced by several subsystems. In such situations, there are two possible ways to handle with this multiplicity:

- the first way is to select the most promising result – i.e., the result for which the estimated quality is the best;
- the second way is, instead of *selecting* one of the results, to *combine* them, hoping that such a combination will avoid random perturbations that make all the results imperfect.

Both ways are used in machine learning, but each one with a twist:

- Instead of selecting the best result, i.e., the result x with the largest value of the corresponding objective function $J(x)$, we select one of the results with some probability. This probability increases with $J(x)$, but remains non-zero even for much smaller values $J(x)$. In other words, instead of "hard maximum", we use "soft maximum" (softmax, for short). Empirically, the most efficient softmax is when the probability of selecting x is proportional to $\exp(\beta \cdot J(x))$ for some $\beta > 0$. A natural question is: why?
- For combination ("averaging"), instead of a seemingly natural arithmetic average, it turns out that the geometric mean often works much better. Why?

What we do in this paper. From the theoretical viewpoint, we have a challenge: for deep learning to become a success, four ideas are needed (and several others as well, we just mentioned the main ones). However, it is not clear why these particular heuristics were successful, and others, seemingly more natural and promising ones, did not work as well.

In this chapter, we provide a theoretical explanation for these empirical successes. It turns out that all these successes can be explained if we apply optimization under uncertainty.

2 Why Rectified Linear Neurons Are Efficient: A Theoretical Explanation

Question: reminder. Let us start with the very first question: which activation function $s_0(z)$ should we use? To be more precise, what is the *optimal* choice of the activation function? Which functions are the best according to some optimality criterion?

To answer this question, let us first recall what we mean by an optimality criterion in the first place. This recall will be useful when answering all other questions as well, so we will formulate it in general terms.

What do we mean by optimality criterion? In general, what do we mean by an optimality criterion, i.e., by a criterion that allows us to select one of many possible alternatives? In many cases, we have a well-defined objective function $F(a)$ – i.e., we have a numerical value $F(a)$ attached to each alternative a. We then select the alternative a for which this value is – depending on what we want – either the largest or the smallest.

For example, when we look for the shortest path:

- we assign, to each path a, its length $F(a)$, and
- we select the path for which this length is the smallest possible.

When we look for an algorithm for solving problems of given size, often:

- we assign, to each algorithm a, the worst-case computation time $F(a)$ on all inputs of this size, and
- we select the algorithm a for which this worst-case time $F(a)$ is the smallest possible.

However, an optimality criterion can be more complicated. For example, we may have several different shortest paths a for a car to go from one city location to another. In this case, it may be reasonable to select, among these shortest paths, a path a along which the overall exposure to pollution $G(a)$ is the smallest. The resulting optimality criterion can no longer be described by a single objective function, it is more complicated: we prefer a to a' if:

- either $F(a) < F(a')$
- or $F(a) = F(a')$ and $G(a) < G(a')$.

Similarly, if we have two different algorithms a with the same worst-case computation time $F(a)$, we may want to select, among them, the one for which the average computation time $G(a)$ is the smallest possible. In this case too, we prefer a to a' if:

- either $F(a) < F(a')$,
- or $F(a) = F(a')$ and $G(a) < G(a')$.

The optimality criterion can be even more complicated. However, no matter how many different objective functions we use, we do need to have a way to compare different alternatives. Thus, we can define a general optimality criterion as a relation $\preceq$ on the set of all possible alternatives, so that $a \preceq a'$ means that the alternative a' is better (or of the same quality) than the alternative a.

Naturally, each alternative has the same quality as itself, so we should have $a \preceq a$. Such relations are called *reflexive*. Also, if a' is better than or of the same

quality as a and a'' is better than or of the same quality as a', then clearly a'' should be better than or of the same quality as a. relations with this property are known as *transitive*. So, we arrive at the following definition.

Definition 1. *Let a set A be given; its elements will be called alternatives.*

- *We say that a relation $\preceq$ on the set A is* reflexive *if $a \preceq a$ for all a.*
- *We say that a relation $\preceq$ is* transitive *if $a \preceq a'$ and $a' \preceq a''$ imply that $a \preceq a''$.*
- *By an* optimality criterion *in the set A, we mean a reflexive transitive relation $\preceq$ on this set.*

What is an optimal alternative? In these terms, an alternative $a_{\rm opt}$ is *optimal* if is better (or of the same quality) than all other alternatives.

Definition 2. *We say that an alternative $a_{\rm opt}$ is* optimal *with respect to the optimality criterion $\preceq$ on the set A if $a \preceq a_{\rm opt}$ for all $a \in A$.*

The optimality criterion must be useful. We want an optimality criterion to be useful, i.e., we want to use it to select an alternative. Thus, there should be at least one alternative which is optimal according to this criterion.

Definition 3. *We say that an optimality criterion $\preceq$ is* useful *if there exists at least one optimal alternative.*

When is the optimality criterion final? What if several different alternatives are optimal according to the given criterion? In this case, we can use this non-uniqueness to optimize something else.

For example, if on a given class of benchmarks, neurons that use several different activation functions have the same average approximation error, we can select, among the, the function with the smallest computational complexity.

This way, instead of the original optimality criterion, we, in effect, use a new criterion according to which s_0 is better than s_0 if:

- either it has the smaller average approximation error
- or it has the same average approximation error and smaller computational complexity.

If, based on this modified criterion, we still have several different activation functions which are equally good, we can use this non-uniqueness to optimize something else: e.g., worse-case approximation accuracy, etc.

Thus, every time the optimality criterion selects several equally good alternatives, we, in effect, replace it with a modified criterion, and keep modifying it until *finally* we get a criterion for which only one alternative is optimal. So, we arrive at the following definition.

Definition 4. *We say that a useful optimality criterion is* final *if there exists exactly one alternative which is optimal with respect to this optimality criterion.*

Natural symmetries: general description. In many practical situations, we have transformations that do not affect the physics of the situation.

For example, when we measure a physical signal, the resulting numerical value depends on what measuring unit we use in this measurement. When we measure the height in meters, the person's height is 1.7. However, if we measure the same height in centimeters, we get a different numerical value: 170.

In general, if instead of the original measuring unit, we use a different unit which us λ times smaller than the previous one, then all the numerical values get multiplied by λ: $x \to \lambda \cdot x$. For example, if we replace meters by centimeters, all numerical values get multiplied by $\lambda = 100$.

If we use a different unit, then numerical values change, but the physical situation remains the same. So, it is reasonable to expect that this change should not affect which alternatives – e.g., which activation functions – should be better and which should be worse.

Other possible transformations come from the fact that for many physical quantities such as time, the choice of a starting point is also arbitrary – we can select a moment of time which is t_0 moments earlier, in which case each numerical value t is replaced by $t + t_0$.

In all such cases, we have a class of corresponding "natural" transformations. Clearly, if two transformation are natural, then their composition – when we first apply the first one and then the second one – should also be natural. Similarly, the inverse should be natural. The class of all transformations with this property is known as a *transformation group*.

Definition 5. *Let a set A be a given.*

- *A* transformation *is a reversible function from A to A.*
- *A set G of transformations is called a* transformation group *if it satisfies the following two properties:*
  - *if $g \in G$, then the inverse transformation g^{-1} should also belong to G, and*
  - *if f and g belong to G, then their composition $f \circ g$ also belongs to G.*
- *Transformations from the group G are called* symmetries.

For natural transformations, the relation $a \preceq a'$ should not change if we apply the same symmetry to both a and a'. This leads to the following definition.

Definition 6. *We say that the optimality criterion $\preceq$ is* invariant *with respect to the transformation group G (or simply G-*invariant*, for short) if for all $a, a' \in A$ and for all $g \in G$, we have*

$$a \preceq a' \Leftrightarrow g(a) \preceq g(a').$$

Main lemma. It turns out that for invariant final optimality criteria, the optimal alternative is also invariant, in some reasonable sense.

Definition 7. *We say that an alternative $a_0 \in A$ is* invariant *with respect to a transformation group G (or, for short, G-*invariant*) if $g(a_0) = a_0$ for all $g \in G$.*

Proposition 1. *For each G-invariant final optimality criterion, the optimal alternative $a_{\rm opt}$ is also G-invariant.*

Comments.

- It is important to emphasize that our result is *not* based on selecting a *single* optimality criterion: it holds for *all* optimality criteria that satisfy reasonable properties – such as being final and being G-invariant.
- For reader's convenience, all the proofs are placed in the special (last) section of this paper.

Natural symmetries of an activation function. We claim that for an activation function, it is important to take into account that we can change the measuring unit and thus, get different numerical values for describing the same quantity.

In the neural networks, inputs are usually normalized, so, at first glance, there seems to be no need to such re-scaling $x \to \lambda \cdot x$. However, normalization parameters may change if we get new data.

For example, often, normalization means that the range of possible values of some positive quantity is linearly re-scaled to the interval $[0, 1]$ – by dividing all inputs by the largest possible value of the corresponding quantity. When we add more data points, we may get values which are somewhat larger than the largest of the previously observed value. In this case, the normalization based on the enlarged data set leads to re-scaling of all previously normalized values – i.e., in effect, to a change in the measuring unit.

It is therefore reasonable to require that the quality of an activation function does not depend on the choice of the measuring unit. Let us describe this property in precise terms.

Scaling-invariance: towards a precise description. Suppose that in some selected units, the activation function has the form $s(x)$. If we replace the original measuring unit by a new unit which is λ times larger that the original one, then the value x in the new units is equivalent to $\lambda \cdot x$ in the old units. If we apply the old-unit activation function to this amount, we get the output of $s(\lambda \cdot x)$ of old units – which is equivalent to $\lambda^{-1} \cdot s(\lambda \cdot x)$ new units.

Thus, after the change in units, the transformation described, in the original units, by an activation function $s(x)$ is described, in the new units, by a modified activation function $\lambda^{-1} \cdot s(\lambda \cdot x)$. So, the above requirement takes the following form:

Definition 8.

- *For each $\lambda > 0$, by a* λ-scaling T_λ, *we mean a transformation from the original function $s(x)$ to a new function $(T_\lambda(s))(x) \stackrel{\text{def}}{=} \lambda^{-1} \cdot s(\lambda \cdot x)$.*
- *We say that an optimality criterion or an alternative are* scale-invariant *if they are invariant with respect to all λ-scalings.*

Now, we are ready to formulate our first result.

Proposition 2. *If a function $s_0(x)$ is optimal with respect to some final scale-invariant optimality criterion, then it has the following form:*

- $s_0(x) = c_+ \cdot x$ *for* $x \geq 0$ *and*
- $s_0(x) = c_- \cdot x$ *for* $x < 0$.

Comment. One can easily check that each such function has the form

$$s_0(x) = c_- \cdot x + (c_+ - c_-) \cdot \max(x, 0).$$

Thus, if $c_+ \neq c_-$, i.e., if the corresponding activation function is not linear, then the class of functions represented by s_0-neural networks coincides with the class of functions represented by rectified linear neural networks.

So, *we have a theoretical justification for the success of rectified linear activation functions.*

Historical comment. This result first appeared in [6].

3 Why Sigmoid Activation Functions

Scalings, shifts, what else? In our explanation of success of rectified linear activation functions, we used scale-invariance. To explain why it is efficient to use the sigmoid activation functions for some layers, let us analyze what other types of invariance we can have. We have already mentioned that we can also have *shift-invariance* $t \to t + t_0$. Together, shifts and scalings form a group of all linear transformations?

Since we want to be able to represent transformations in a computer, and in a computer, we can only store finitely many parameter values, the desired group should depend only on a finite number of parameters – i.e., it should be *finite-dimensional*. It turns out that the only finite-dimensional group that strictly contains the group of all linear transformations is the group of all fractionally linear transformations $g(x) = \dfrac{A \cdot x + B}{C \cdot x + D}$. The proof of this result can be found, e.g., in [13, 15, 18] (a more general result was formulated by Norbert Wiener, the father of cybernetics, in [19]). Under assumptions of smoothness, the proof is given in the proofs section.

We should consider families of functions. If we have a reasonable transformation g and $x \to y = s_0(x)$ is a reasonable activation function, then after re-scaling $y \to g(y)$, we shall also get a reasonable activation function. Thus, with

the original function $s_0(z)$, all the functions $g(s_0(z))$ should also be reasonable. From this viewpoint, instead of considering the original activation functions, it takes more sense to consider *families* of such functions $\{g(s_0(x))\}_{g\in G}$. Out of all such families, we should select the optimal one.

In addition to changing the result y of neural activity, we can also re-scale the inputs x – e.g., by shift, which corresponds to changing the starting point for x. It is reasonable to require that the relative quality of different families of activation functions should not depend on what starting point we use. Let us describe this in precise terms.

Definition 9. *By a* family *of activation functions, we mean the family of all the functions of the type* $\dfrac{A\cdot s_0(x)+B}{C\cdot s_0(x)+D}$, *where* $s_0(x)$ *is a given smooth non-decreasing function and* A, B, C, *and* D *are arbitrary constants.*

Definition 10. *By a* shift, *we means a transformation* $s_0(x)\to s_0(x+x_0)$ *for some* x_0.

Proposition 3. *For every shift-invariant final optimality criterion on the set of all families of activation functions, the optimal family corresponds to* $s_0(x)=x$, $s_0(x)=\exp(\beta\cdot x)$, *or to the sigmoid function* $s_0(x)=\dfrac{1}{1+c\cdot\exp(-\beta\cdot x)}$.

Comment. The sigmoid function can be reduced to its standard form by an appropriate re-scaling of x: from x to $x'=\beta\cdot x-\ln(c)$. The cases $s_0(x)=x$ and $s_0(x)=\exp(\beta\cdot x)$ are actually limit cases of the sigmoid. Thus, the above result indeed explains the empirical optimality of the sigmoid activation function.

Historical comment. This result was previously mentioned in [12, 13, 15, 18].

4 Selection of Poolings

What is pooling: towards a precise definition. We start with m values $a_1,\dots,a_m$, and we want to generate a single value a that represents all these values.

In the case of arithmetic average, pooling means that we select a value a for which $a_1+\ldots+a_m=a+\ldots+a$ (m times). In general, pooling means that we select some combination operation $*$ and we then select the value a for which $a_1*\ldots*a_m=a*\ldots*a$ (m times). For example, if, as a combination operation, we select $\max(a,b)$, then the corresponding condition

$$\max(a_1,\dots,a_n)=\max(a,\dots,a)=a$$

describes the max-pooling.

From this viewpoint, selecting pooling means selecting an appropriate combination operation. Thus, selecting the optimal pooling means selecting the optimal combination operation.

Natural properties of a combination operation. The combination operation transforms two non-negative values – such as intensity of an image at a

given location – into a single non-negative value. The result of applying this operation should not depend on the order in which we combine the values. Thus, we should have $a * b = b * a$ (commutativity) and $a * (b * c) = (a * b) * c$ (associativity).

Definition 11. *By a* combination operation*, we mean a commutative, associative operation $a * b$ that transforms two non-negative real numbers a and b into a non-negative real number $a * b$.*

Scale-invariance. As we have mentioned earlier, numerical values of a physical quantity depend on the choice of a measuring unit. It is therefore reasonable to require that the preference relation should not change if we simply change the measuring unit. Let us describe this requirement in precise terms. If, in the original units, we had the operation $a * b$, then, in the new units, the operation will take the following form:

- first, we transform the value a and b into the new units, so we get $a' = \lambda \cdot a$ and $b' = \lambda \cdot b$;
- then, we combine the new numerical values, getting $(\lambda \cdot a) * (\lambda \cdot b)$;
- finally, we re-scale the result to the original units, getting $R_\lambda(*)$ defined as
$$aR_\lambda(*)b \stackrel{\text{def}}{=} \lambda^{-1} \cdot ((\lambda \cdot a) * (\lambda \cdot b).$$

It therefore makes sense to require that is $* \preceq *'$, then for every $\lambda > 0$, we get $R_\lambda(*) \preceq R_\lambda(*')$.

Definition 12. *We say that an optimality criterion on the set of all combination operations is* scale-invariant *if for all $\lambda > 0$, $* \preceq *'$ implies $R_\lambda(*) \preceq R_\lambda(*')$, where $aR_\lambda(*)b \stackrel{\text{def}}{=} \lambda^{-1} \cdot ((\lambda \cdot a) * (\lambda \cdot b))$.*

Shift-invariance. The numerical values also change if we change the starting point for measurements. For example, when measuring intensity, we can measure the actual intensity of an image, or we can take into account that there is always some noise $a_0 > 0$, and use the noise-only level a_0 as the new starting point. In this case, instead of each original value a, we get a new numerical value $a' = a - a_0$ for describing the same physical quantity.

If we apply the combination operation in the new units, then in the old units, we get a slightly different result; namely,

- first, we transform the value a and b into the new units, so we get $a' = a - a_0$ and $b' = b - a_0$;
- then, we combine the new numerical values, getting $(a - a_0) * (b - a_0)$;
- finally, we re-scale the result to the original units, getting $S_{a_0}(*)$ defined as
$$aS_{a_0}(*)b \stackrel{\text{def}}{=} (a - a_0) * (b - a_0) + a_0.$$

It makes sense to require that the preference relation not change if we simply change the starting point: so if $* \preceq *'$, then for every a_0, we get $S_{a_0}(*) \preceq S_{a_0}(*')$.

Definition 13. *We say that an optimality criterion is* shift-invariant *if for all a_0, $* \preceq *'$ implies $S_{a_0}(*) \preceq S_{a_0}(*')$, where $aS_{a_0}(*)b \stackrel{\text{def}}{=} ((a - a_0) * (b - a_0)) + a_0$.*

Weak version of shift-invariance. Alternatively, we can have a weaker version of this "shift-invariance" if we require that shifts in a and b imply a possibly different shift in $a * b$, i.e., if we shift both a and b by a_0, then the value $a * b$ is shifted by some value $f(a_0)$ which is, in general, different from a_0.

Definition 14. *We say that an optimality criterion is* weakly shift-invariant *if for every a_0, there exists a value $f(a_0)$ such that $* \preceq *'$ implies $W_{a_0}(*) \preceq W_{a_0}(*')$, where*

$$aW_{a_0}(*)b \stackrel{\text{def}}{=} ((a - a_0) * (b - a_0)) + f(a_0).$$

Now, we are ready to formulation our results.

Proposition 4. *For every final, scale- and shift-invariant optimality criterion, the optimal combination operation has one of the following two forms: $a * b = \min(a, b)$ or $a * b = \max(a, b)$.*

Discussion. Since the max combination operation corresponds to max-pooling, *this result explains why max-pooling is empirically the best combination operation.*

Proposition 5. *For every final, scale-invariant and weakly shift-invariant optimality criterion, the optimal combination operation has one of the following four forms: $a * b = 0$, $a * b = \min(a, b)$, $a * b = \max(a, b)$, or $a * b = a + b$.*

Discussion. Since the addition combination operation corresponds to average-based pooling, *this result explains why max-pooling and average-pooling are empirically the best combination operations.*

Historical comment. This result first appeared in [5].

5 Why Softmax

Formulation of the problem. We want to describe how the probability $p(a)$ of selecting an alternative a depends on the value $J(a)$) of the corresponding objective function. In other words, we want to find a non-decreasing function $f(x)$ for which the probability $p(a)$ is proportional to $f(J(a))$, i.e., for which $p(a) = c \cdot f(J(a))$. Since the probabilities add up to 1, we have $\sum\limits_{a \in A} p(a) =$

$c \cdot \sum\limits_{a \in A} f(J(a)) = 1$, hence $c = \dfrac{1}{\sum\limits_{a \in A} f(J(a))}$ and

$$p(a) = \frac{f(J(a))}{\sum\limits_{a' \in A} f(J(a'))}.$$

One can easily see that if we multiply all the values of the function $f(x)$ by the same constant, the probabilities remain the same. Since we are interested only in the probabilities, this means that instead of selecting a single function $f(x)$, we should select a *family* of functions $\{c \cdot f(x)\}_c$, where a function $f(x)$ is given, and c takes all possible positive values.

Definition 15. *By a* family*, we mean a family of functions $\{c \cdot f(x)\}_c$, where $f(x)$ is a given non-decreasing function and c takes all possible positive values.*

What are natural symmetries here? The main purpose of selecting an objective function is to decide which alternative is better and which is worse. If we add a constant to all the values of the objective function, this does not change which ones are better and which ones are worse. Thus, it is reasonable to require that the optimality criterion on the class of all the families should not change if we simply add a constant x_0 to all the values x, i.e., replace each function $f(x)$ with $f(x + x_0)$.

Definition 16. *For each x_0, by a* x_0-shift *of a family $\{c \cdot f(x)\}_c$, we mean the family $\{c \cdot f(x + x_0)\}_c$.*

Proposition 6. *For each final shift-invariant optimality criterion on the set of all families, the optimal family corresponds to $f(x) = \exp(\beta \cdot x)$ for some $\beta \geq 0$.*

Discussion. This result explains why this particular version of softmax has been most empirically successful.

Historical comment. This result, in effect, appeared in [11, 12, 15].

6 Which Averaging Should We Choose

Discussion. Averaging is similar to pooling, with the following main difference:

- in pooling, we combine the original measurement result, with possibly a lot of noise, so noise-related shifts make sense;
- in contrast, in averaging, we combine the results of processing, where the noise has been largely eliminated; so, shift no longer makes sense, only scaling makes sense.

What does make sense here is monotonicity: if we increase one or both numbers, the result should increase – or at least stay the same.

Definition 17. *We say that a combination operation $a * b$ is* monotonic *if whenever $a \le a'$ and $b \le b'$, then $a * b \le a' * b'$.*

Proposition 7. *For every final scale-invariant optimality criterion on the set of all monotonic combination operations, the optimal combination operation has one of the following forms: $a * b = 0$, $a * b = \min(a, b)$, $a * b = \max(a, b)$, and $a * b = (a^\alpha + b^\alpha)^{1/\alpha}$ for some α.*

Discussion. What are the "averaging" operations corresponding to these optimal combination operations?

For $a * b = 0$, the property $v_1 * \ldots * v_m = v * \ldots * v$ is satisfied for any possible v, so this combination operation does not lead to any "averaging" at all.

For $a * b = \min(a, b)$, the condition $v_1 * \ldots * v_m = v * \ldots * v$ leads to

$$v = \min(v_1, \ldots, v_m).$$

For $a * b = \max(a, b)$, the condition $v_1 * \ldots * v_m = v * \ldots * v$ leads to

$$v = \max(v_1, \ldots, v_m).$$

As we have mentioned, this "averaging" operation is actually sometimes used in deep learning – namely, in pooling.

Finally, for the combination operation $a * b = (a^\alpha + b^\alpha)^{1/\alpha}$, the condition $v_1 * \ldots * v_m = v * \ldots * v$ leads to

$$v = \left(\frac{v_1^\alpha + \ldots + v_m^\alpha}{m}\right)^{1/\alpha}.$$

For $\alpha = 1$, we get arithmetic average, and for $\alpha \to 0$, we get the geometric mean – the combination operation which turned out to be empirically the best for deep learning-related dropout training.

Indeed, in this case, the condition $v_1 * \ldots * v_m = v * \ldots * v$ takes the form

$$(v_1^\alpha + \ldots + v_m^\alpha)^{1/\alpha} = (v^\alpha + \ldots + v^\alpha)^{1/\alpha},$$

which is equivalent to

$$v_1^\alpha + \ldots + v_m^\alpha = m \cdot v^\alpha.$$

For every real value a, we have

$$a^\alpha = (\exp(\ln(a))^\alpha = \exp(\alpha \cdot \ln(a)).$$

For small x, $\exp(x) \approx 1 + x$, so $a^\alpha \approx 1 + \alpha \cdot \ln(a)$. Thus, the above condition leads to

$$(1 + \alpha \cdot \ln(v_1)) + \ldots + (1 + \alpha \cdot \ln(v_m)) = m \cdot (1 + \alpha \cdot \ln(v)),$$

i.e., to

$$m + \alpha \cdot (\ln(v_1) + \ldots + \ln(v_m)) = m + m \cdot \alpha \cdot \ln(v),$$

and thus, to

$$\ln(v) = \frac{\ln(v_1) + \ldots + \ln(v_m)}{m} = \frac{\ln(v_1 \cdot \ldots \cdot v_m)}{m};$$

hence to $v = \sqrt[m]{v_1 \cdot \ldots \cdot v_m}$.

So, we indeed have a 1-D family that contains combination operations efficiently used in deep learning:

- the arithmetic average that naturally comes from the use of the Least Squares optimality criterion, and
- the geometric mean, empirically the best combination operation for deep learning-related dropout training.

Historical comment. This result first appeared in [7]; it is based on a theorem proven in [2].

7 Proofs

Proof of Proposition 1. Let $a_{\rm opt}$ be the optimal alternative, and let g be any transformation from the group G. We want to prove that $g(a_{\rm opt}) = a_{\rm opt}$. To prove this equality, let us prove that the alternative $g(a_{\rm opt})$ is optimal. Then, the desired equality will follow from the fact that the optimality criterion is final – and thus, there is only one optimal alternative.

To prove that the alternative $g(a_{\rm opt})$ is optimal, we need to prove that $a \preceq g(a_{\rm opt})$ for all a. Due to G-invariance of the optimality criterion, this condition is equivalent to $g^{-1}(a) \preceq a_{\rm opt}$ – which is, of course, always true, since $a_{\rm opt}$ is optimal. Thus, $g(a_{\rm opt})$ is also optimal, hence $g(a_{\rm opt}) = a_{\rm opt}$ for all $g \in G$. The proposition is proven.

Proof of Proposition 2. According to Proposition 1, the optimal activation function $s_0(x)$ should be scale-invariant. In other words, for all x and all $\lambda > 0$, we have $\lambda^{-1} \cdot s_0(\lambda \cdot x) = s_0(x)$, thus

$$s_0(\lambda \cdot x) = \lambda \cdot s_0(x).$$

Let us show that this property leads to the desired form of the activation function.

Every input x is either equal to 0, or positive, or negative. Let us consider these three cases one by one.

1°. Let us first consider the case of $x = 0$.

For $x = 0$ and $\lambda = 2$, scale invariance means that if $y = s_0(0)$, then $2y = s_0(0)$. Thus, $2y = y$, hence $y = s_0(0) = 0$.

2°. Let us now consider the case of positive values x.

Let us denote $c_+ \stackrel{\rm def}{=} s_0(1)$. Then, by using scale-invariance with:

- x instead of λ,
- 1 instead of x, and
- c_+ instead of $s_0(1)$,

we conclude that for all $x > 0$, we have $s_0(x) = x \cdot c_+$.

For positive values x, the desired equality is proven.

3°. To complete the proof of this result, we need to prove it for negative inputs x.

Let us denote $c_- \stackrel{\text{def}}{=} -s_0(-1)$. In this case, $s_0(-1) = -c$. Thus, for every $x < 0$, by using scale-invariance with:

- $\lambda = |x|$,
- $x = -1$, and
- $s_0(-1) = -c_-$,

we conclude that

$$s_0(x) = s_0(|x| \cdot (-1)) = |x| \cdot s_0(-1) = |x| \cdot (-c_-) = c_- \cdot x.$$

The proposition is proven.

Proof that every transformation from a finite-dimensional group containing all linear transformations if fractionally linear. Every transformation is a composition of infinitesimal ones

$$x \to x + \varepsilon \cdot f(x),$$

for infinitely small ε. So, it's enough to consider infinitesimal transformations.

The class of the corresponding functions $f(x)$ is known as a *Lie algebra* A of the corresponding transformation group. Infinitesimal linear transformations correspond to

$$f(x) = a + b \cdot x,$$

so all linear functions are in A. In particular, $1 \in A$ and $x \in A$.

For any λ, the product $\varepsilon \cdot \lambda$ is also infinitesimal, so we get

$$x \to x + (\varepsilon \cdot \lambda) \cdot f(x) = x \to x + \varepsilon \cdot (\lambda \cdot f(x)).$$

So, if $f(x) \in A$, then $\lambda \cdot f(x) \in A$.

If we first apply $f(x)$, then $g(x)$, we get

$$x \to (x + \varepsilon \cdot f(x)) + \varepsilon \cdot g(x + \varepsilon \cdot f(x)) =$$

$$x + \varepsilon \cdot (f(x) + g(x)) + o(\varepsilon).$$

Thus, if $f(x) \in A$ and $g(x) \in A$, then $f(x) + g(x) \in A$. So, A is a linear space.

In general, for the composition, we get

$$x \to (x + \varepsilon_1 \cdot f(x)) + \varepsilon_2 \cdot g(x_1 + \varepsilon_1 \cdot f(x)) =$$

$$x + \varepsilon_1 \cdot f(x) + \varepsilon_2 \cdot g(x) + \varepsilon_1 \cdot \varepsilon_2 \cdot g'(x) \cdot f(x) + \text{ quadratic terms.}$$

If we then apply the inverses to $x \to x + \varepsilon_1 \cdot f(x)$ and $x \to x + \varepsilon_2 \cdot g(x)$, the linear terms disappear, we get:

$$x \to x + \varepsilon_1 \cdot \varepsilon_2 \cdot \{f, g\}(x),$$

where

$$\{f, g\} \stackrel{\text{def}}{=} f'(x) \cdot g(x) - f(x) \cdot g'(x).$$

Thus, if $f(x) \in A$ and $g(x) \in A$, then $\{f, g\}(x) \in A$. The expression $\{f, g\}$ is known as the *Poisson bracket*.

Let's expand any function $f(x)$ in Taylor series:

$$f(x) = a_0 + a_1 \cdot x + \ldots$$

If k is the first non-zero term in this expansion, we get

$$f(x) = a_k \cdot x^k + a_{k+1} \cdot x^{k+1} + a_{k+2} \cdot x^{k+2} + \ldots$$

For every λ, the algebra A also contains

$$\lambda^{-k} \cdot f(\lambda \cdot x) = a_k \cdot x^k + \lambda \cdot a_{k+1} \cdot x^{k+1} + \lambda^2 \cdot a_{k+2} \cdot x^{k+2} + \ldots$$

In the limit $\lambda \to 0$, we get $a_k \cdot x^k \in A$, hence $x^k \in A$.

Thus,

$$f(x) - a_k \cdot x^k = a_{k+1} \cdot x^{k+1} + \ldots \in A.$$

We can similarly conclude that A contains all the terms x^n for which $a_n \neq 0$ in the original Taylor expansion.

Since $g(x) = 1 \in A$, for each $f \in A$, we have

$$\{f, 1\} = f'(x) \cdot 1 + f(x) \cdot q' = f'(x) \in A.$$

Thus, for each k, if $x^k \in A$, we have

$$(x^k)' = k \cdot x^{k-1} \in A$$

hence $x^{k-1} \in A$, etc. So, if $x^k \in A$, all smaller power are in A too. In particular, this means that if $x^k \in A$ for some $k \geq 3$, then we have $x^3 \in A$ and $x^2 \in A$; thus:

$$\{x^3, x^2\} = (x^3)' \cdot x^2 - x^3 \cdot (x^2)' = 3 \cdot x^2 \cdot x^2 - x^3 \cdot 2 \cdot x = x^4 \in A.$$

In general, once $x^k \in A$ for $k \geq 3$, we get

$$\{x^k, x^2\} = (x^k)' \cdot x^2 - x^k \cdot (x^2)' = k \cdot x^{k-1} \cdot x^2 - x^k \cdot 2 \cdot x =$$

$$(k-2) \cdot x^{k+1} \in A \text{ hence } x^{k+1} \in A.$$

So, by induction, $x^k \in A$ for all k. Thus, A is infinite-dimensional – which contradicts to our assumption that A is finite-dimensional.

So, we cannot have Taylor terms of power $k \geq 3$; therefore we have:

$$x \to x + \varepsilon \cdot (a_0 + a_1 \cdot x + a_2 \cdot x^2).$$

This corresponds to an infinitesimal fractional-linear transformation

$$x \to \frac{\varepsilon \cdot A + (1 + \varepsilon \cdot B) \cdot x}{1 + \varepsilon \cdot D \cdot x} =$$

$$(\varepsilon \cdot A + (1 + \varepsilon \cdot B) \cdot x) \cdot (1 - \varepsilon \cdot D \cdot x) + o(\varepsilon) =$$

$$x + \varepsilon \cdot (A + (B - D) \cdot x - D \cdot x^2).$$

So, to match, we need

$$A = a_0, \quad D = -a_2, \text{ and } B = a_1 - a_2.$$

We concluded that every infinitesimal transformation is fractionally linear. Every transformation is a composition of infinitesimal ones. Composition of fractional-linear transformations is fractional linear. Thus, all transformations are fractional linear.

Proof of Proposition 3. Due to Proposition 1, we can conclude that the optimal family is also shift-invariance, i.e., that for each c, the function $s_0(x-c)$ also belongs to the optimal family. Thus, we conclude that for every x, there exists values A, B, C, and D (which depend on c) for which

$$s_0(x - c) = \frac{A(c) \cdot s_0(x) + B(c)}{C(c) \cdot s_0(x) + D(c)}.$$

For $c = 0$, we get $A(0) = D(0) = 1$, $B(0) = C(0) = 0$. Differentiating the above equation by c and taking $c = 0$, we get a differential equation for $s_0(x)$:

$$-\frac{ds_0}{dx} = (A'(0) \cdot s_0(x) + B'(0)) - s_0(x) \cdot (C'(0) \cdot s_0(x) + D'(0)).$$

So,

$$\frac{ds_0}{C'(0) \cdot s_0^2 + (A'(0) - C'(0)) \cdot s_0 + B'(0)} = -dx.$$

Integrating and taking into account that the activation function must be non-decreasing, we indeed get the desired expressions (after an appropriate linear re-scaling of $s_0(x)$). The proposition is proven.

Proof of Proposition 4.

1°. Let $a * b$ be the optimal combination operation. Due to Proposition 1, this operation is itself scale-invariant and shift-invariant. Let us prove that it has one of the above two forms.

For every pair (a, b), we can have three different cases: $a = b$, $a < b$, and $a > b$. Let us consider them one by one.

2°. Let us first consider the case when $a = b$.

Let us denote $v \stackrel{\text{def}}{=} 1 * 1$. From scale-invariance with $\lambda = 2$, from $1 * 1 = v$, we get $2 * 2 = 2v$. From shift-invariance with $s = 1$, from $1 * 1 = v$, we get $2 * 2 = v + 1$. Thus, $2v = v + 1$, hence $v = 1$, and $1 * 1 = 1$.

For $a > 0$, by applying scale-invariance with $\lambda = a$ to the formula $1 * 1 = 1$, we get $a * a = a$.

For $a = 0$, if we denote $c \stackrel{\text{def}}{=} 0 * 0$, then, by applying shift-invariance with $s = 1$ to $0 * 0 = c$, we get $1 * 1 = c + 1$. Since we already know that $1 * 1 = 1$, this means that $c + 1 = 1$ and thus, that $c = 0$, i.e., that $0 * 0 = 0$.

So, for all $a \geq 0$, we have $a * a = a$. In this case, $\min(a, a) = \max(a, a) = a$, so we have $a * a = \min(a, a)$ and $a * a = \max(a, a)$.

3°. Let us first consider the case when $a < b$. In this case, $b - a > 0$.

Let us denote $t \stackrel{\text{def}}{=} 0 * 1$. By applying scale-invariance with $\lambda = b - a > 0$ to the formula $0 * 1 = t$, we conclude that

$$0 * (b - a) = (b - a) \cdot t. \tag{1}$$

Now, by applying shift-invariance with $s = a$ to the formula (1), we conclude that

$$a * b = (b - a) \cdot t + a. \tag{2}$$

To find possible values of t, let us take into account that the combination operation should be associative. This means, in particular, that for all possible triples a, b, and c for which we have $a < b < c$, we must have

$$a * (b * c) = (a * b) * c. \tag{3}$$

Since $b < c$, by the formula (2), we have

$$b * c = (c - b) * t + b.$$

Since $t \geq 0$, we have $b * c \geq b$ and thus, $a < b * c$. So, to compute $a * (b * c)$, we can also use the formula (2), and get

$$a * (b * c) = (b * c - a) \cdot t + a = ((c - b) \cdot t + b) \cdot t + a = c \cdot t^2 + b \cdot (t - t^2) + a. \tag{4}$$

Let us restrict ourselves to the case when $a * b < c$. In this case, the general formula (2) implies that

$$(a * b) * c = (c - a * b) \cdot t + a * b = (c - ((b - a) \cdot t + a)) \cdot t + (b - a) \cdot t + a,$$

so

$$(a * b) * c = c \cdot t + b \cdot (t - t^2) + a \cdot (1 - t)^2. \tag{5}$$

Due to associativity, formulas (4) and (5) must coincide for all a, b, and c for which $a < b < c$ and $c > a * b$. Since these two linear expressions must be equal for all sufficiently large values of c, the coefficients at c must be equal, i.e., we

must have $t = t^2$. From $t = t^2$, we conclude that $t - t^2 = t \cdot (1 - t) = 0$, so either $t = 0$ or $1 - t = 0$ (in which case $t = 1$).

If $t = 0$, then the formula (2) has the form $a * b = a$, i.e., since $a < b$, the form $a * b = \min(a, b)$.

If $t = 1$, then the formula (2) has the form $a * b = (b - a) + a = b$, i.e., since $a < b$, the form $a * b = \max(a, b)$.

4°. If $a > b$, then, by commutativity, we have $a * b = b * a$, where now $b < a$. So:

- if $t = 0$, then, due to Part 3 of this proof, we have $b * a = \min(b, a)$; since $a * b = b * a$ and since clearly $\min(a, b) = \min(b, a)$, we can conclude that $a * b = \min(a, b)$ for $a > b$ as well;
- if $t = 1$, then, due to Part 3 of this proof, we have $b * a = \max(b, a)$; since $a * b = b * a$ and since clearly $\max(a, b) = \max(b, a)$, we can conclude that $a * b = \max(a, b)$ for $a > b$ as well.

So, either we have $a * b = \min(a, b)$ for all a and b, or we have $a * b = \max(a, b)$ for all a and b. The proposition is proven.

Proof of Proposition 5.

1°. Let $a * b$ be the optimal combination operation. Due to Proposition 1, this operation is scale-invariant and weakly shift-invariant – which means that $a * b = c$ implies $(a + s) * (b + s) = c + f(s)$. Let us prove that the optimal operation $*$ has one of the above four forms.

1°. Let us first prove that $0 * 0 = 0$.

Indeed, let s denote $0 * 0$. Due to scale-invariance, $0 * 0 = s$ implies that $(2 \cdot 0) * (2 \cdot 0) = 2s$, i.e., that $0 * 0 = 2s$. So, we have $s = 2s$, hence $s = 0$ and $0 * 0 = 0$.

2°. Similarly, if we denote $v \stackrel{\text{def}}{=} 1 * 1$, then, due to scale-invariance with $\lambda = a$, $1 * 1 = v$ implies that $a * a = v \cdot a$ for all a.

On the other hand, due to weak shift-invariance with $a_0 = a$, $0 * 0 = 0$ implies that $a * a = f(a)$. Thus, we conclude that $f(a) = v \cdot a$.

2°. Let us now consider the case when $a < b$ and, thus, $b - a > 0$.

Let us denote $t \stackrel{\text{def}}{=} 0 * 1$. From scale-invariance with $\lambda = b - a$, from $0 * 1 = t \geq 0$, we get $0 * (b - a) = t \cdot (b - a)$. From weak shift-invariance with $a_0 = a$, we get $a * b = t \cdot (b - a) + v \cdot a$, i.e.,

$$a * b = t \cdot b + (v - t) \cdot a. \tag{6}$$

Similarly to the proof of Proposition 1, to find possible values of t, let us take into account that the combination operation should be associative. This means, in particular, that for all possible triples a, b, and c for which we have $a < b < c$, we must have

$$a * (b * c) = (a * b) * c.$$

Since $b < c$, by the formula (6), we have

$$b * c = t \cdot c + (v - t) \cdot b.$$

3°. We know that $t \geq 0$. This means that we have either $t > 0$ and $t = 0$.

4°. Let us first consider the case when $t > 0$. In this case, for sufficiently large c, we have $b * c > a$. So, by applying the formula (6) to a and $b * c$, we conclude that

$$a * (b * c) = t \cdot (b * c) + (v - t) \cdot a = t^2 \cdot c + t \cdot (v - t) \cdot b + (v - t) \cdot a. \qquad (7)$$

For sufficient large c, we also have $a * b < c$. In this case, the general formula (6) implies that

$$(a * b) * c = (t \cdot b + (v - t) \cdot a) * c = t \cdot c + t \cdot (v - t) \cdot b + (v - t)^2 \cdot a. \qquad (8)$$

Due to associativity, formulas (7) and (8) must coincide for all a, b, and c for which $a < b < c$, $c > a * b$, and $b * c > a$. Since these two linear expressions must be equal for all sufficiently large values of c, the coefficients at c must be equal, i.e., we must have $t = t^2$.

From $t = t^2$, we conclude that $t - t^2 = t \cdot (1 - t) = 0$. Since we assumed that $t > 0$, we must have $t - 1 = 0$, i.e., $t = 1$.

The coefficients at a must also coincide, so we must have $v - t = (v - t)^2$, hence either $v - t = 0$ or $v - t = 1$. In the first case, the formula (6) becomes $a * b = b$, i.e., $a * b = \max(a, b)$ for all $a \leq b$. Since the operation $*$ is commutative, this equality is also true for $b \leq a$ and is, thus, true for all a and b.

In the second case, the formula (6) becomes $a * b = a + b$ for all $a \leq b$. Due to commutativity, this formula holds for all a and b.

5°. Let us now consider the case when $t = 0$. In this case, the formula (6) takes the form $a * b = (v - t) \cdot a$.

Here, $a * b \geq 0$, thus $v - t \geq 0$. If $v - t = 0$, this implies that $a * b = 0$ for all $a \leq b$ and thus, due to commutativity, for all a and b.

Let us now consider the remaining case when $v - t > 0$. In this case, if $a < b < c$, then for sufficiently large c, we have $a * b < c$, hence

$$(a * b) * c = (v - t) \cdot (a * b) = (v - t) \cdot ((v - t) \cdot a) = (v - t)^2 \cdot a.$$

On the other hand, here $b * c = (v - t) \cdot b$. So, for sufficiently large b, we have $(v - t) \cdot b > a$, thus

$$a * (b * c) = (v - t) \cdot a.$$

Due to associativity, we have $(v - t)^2 \cdot a = (v - t) \cdot a$, hence $(v - t)^2 = v - t$ and, since $v - t > 0$, we have $v - t = 1$. In this case, the formula (6) takes the form $a * b = a = \min(a, b)$ for all $a \leq b$. Thus, due to commutativity, we have $a * b = \min(a, b)$ for all a and b.

We have thus shown that the combination operation indeed has one of the four forms. Proposition 5 is therefore proven.

Proof of Proposition 6. Due to Proposition 1, the optimal family should be shift-invariant. This means, in particular, that if we take the function $f(x)$ from the original optimal family and shift it, then the resulting function $f(x + x_0)$ should also belong to the same optimal family, i.e., we should have $f(x + x_0) = c(x_0) \cdot f(x)$ for some c depending on x_0.

It is known that the only non-decreasing solutions to this functional equation are functions $f(x) = \text{const} \cdot \exp(\beta \cdot x)$ for some $\beta > 0$; see, e.g., [1]. The proposition is proven.

Proof of Proposition 7. Due to Proposition 1, the optimal combination operation should be scale-invariant, i.e., we should have $(\lambda \cdot a) * (\lambda \cdot b) = \lambda \cdot (a * b)$ for all $\lambda > 0$, a, and b. To avoid confusion, let us denote $a * b$ by $f(a, b)$.

1°. Depending on whether the value $f(1, 1)$ is equal to 1 or not, we have two possible cases: $f(1, 1) = 1$ and when $f(1, 1) \neq 1$. Let us consider these two cases one by one.

2°. Let us first consider the case when $f(1, 1) = 1$. In this case, the value $f(0, 1)$ can be either equal to 0 or different from 0. Let us consider both subcases.

2.1°. Let us first consider the first subcase, when $f(0, 1) = 0$.

In this case, for every $b > 0$, scale invariance with $\lambda = b$ implies that

$$f(b \cdot 0, b \cdot 1) = b \cdot 0,$$

i.e., that $f(0, b) = 0$. By taking $b \to 0$ and using continuity, we also get $f(0, 0) = 0$. Thus, $f(0, b) = 0$ for all b.

By commutativity, we have $f(a, 0) = 0$ for all a. So, to fully describe the operation $f(a, b)$, it is sufficient to consider the cases when $a > 0$ and $b > 0$.

$2.1.1^\circ$. Let us prove, by contradiction, that in this subcase, we have $f(1, a) \leq 1$ for all a.

Indeed, let us assume that for some a, we have $b \stackrel{\text{def}}{=} f(1, a) > 1$. Then, due to associativity and $f(1, 1) = 1$, we have $f(1, b) = f(1, f(1, a)) = f(f(1, 1), a) = f(1, a) = b$.

Due to scale-invariance with $\lambda = b$, the equality $f(1, b) = b$ implies that $f(b, b^2) = b^2$. Thus, $f(1, b^2) = f(1, f(b, b^2)) = f(f(1, b), b^2) = f(b, b^2) = b^2$.

Similarly, from $f(1, b^2) = b^2$, we conclude that for $b^4 = (b^2)^2$, we have $f(1, b^4) = b^4$, and, in general, that $f(1, b^{2^n}) = b^{2^n}$ for every n.

Scale invariance with $\lambda = b^{-2^n}$ implies that $f(b^{-2^n}, 1) = 1$. In the limit $n \to \infty$, we get $f(0, 1) = 1$, which contradicts to our assumption that $f(0, 1) = 0$. This contradiction shows that indeed, $f(1, a) \leq 1$.

$2.1.2^\circ$. For $a \geq 1$, monotonicity implies $1 = f(1, 1) \leq f(1, a)$, so $f(1, a) \leq 1$ implies that $f(1, a) = 1$.

Now, for any a' and b' for which $0 < a' \leq b'$, if we denote $r \stackrel{\text{def}}{=} \dfrac{b'}{a'} \geq 1$, then scale-invariance with $\lambda = a'$ implies that $a' \cdot f(1, r) = f(a' \cdot 1, a' \cdot r) = f(a', b')$.

Here, $f(1,r)=1$, thus $f(a',b')=a'\cdot 1=a'$, i.e., $f(a',b')=\min(a',b')$. Due to commutativity, the same formula also holds when $a'\geq b'$. So, in this case, $f(a,b)=\min(a,b)$ for all a and b.

2.2°. Let us now consider the second subcase of the first case, when $f(0,1)>0$.

2.2.1°. Let us first show that in this subcase, we have $f(0,0)=0$.

Indeed, scale-invariance with $\lambda=2$ implies that from $f(0,0)=a$, we can conclude that $f(2\cdot 0,2\cdot 0)=f(0,0)=2\cdot a$. Thus $a=2\cdot a$, hence $a=0$. The statement is proven.

2.2.2°. Let us now prove that in this subcase, $f(0,1)=1$.

Indeed, in this case, for $a\stackrel{\text{def}}{=}f(0,1)$, we have, due to $f(0,0)=0$ and associativity, that $f(0,a)=f(0,f(0,1))=f(f(0,0),1)=f(0,1)=a$. Here, $a>0$, so by applying scale invariance with $\lambda=a^{-1}$, we conclude that $f(0,1)=1$.

2.2.3°. Let us now prove that for every $a\leq b$, we have $f(a,b)=b$. So, due to commutativity, we have $f(a,b)=\max(a,b)$ for all a and b.

Indeed, from $f(1,1)=1$ and $f(0,1)=1$, due to scale invariance with $\lambda=b$, we conclude that $f(0,b)=b$ and $f(b,b)=b$. Due to monotonicity, $0\leq a\leq b$ implies that $b=f(0,b)\leq f(a,b)\leq f(b,b)=b$, thus $f(a,b)=b$. The statement is proven.

3°. Let us now consider the remaining case when $f(1,1)\neq 1$.

3.1°. Let us denote $v(k)\stackrel{\text{def}}{=}f(1,f(\ldots,1)\ldots)$ (k times). Then, due to associativity, for every m and n, the value $v(m\cdot n)=f(1,f(\ldots,1)\ldots)$ ($m\cdot n$ times) can be represented as

$$f(f(1,f(\ldots,1)\ldots),\ldots,f(1,f(\ldots,1)\ldots)),$$

where we divide the 1s into m groups with n 1s in each. For each group, we have $f(1,f(\ldots,1)\ldots)=v(n)$. Thus, $v(m\cdot n)=f(v(n),f(\ldots,v(n))\ldots)$ (m times).

We know that $f(1,f(\ldots,1)\ldots)$ (m times) $=v(m)$. Thus, by using scale-invariance with $\lambda=v(n)$, we conclude that $v(m\cdot n)=v(m)\cdot v(n)$, i.e., that the function $v(n)$ is multiplicative. In particular, this means that for every number p and for every positive integer n, we have $v(p^n)=(v(p))^n$.

3.2°. If $v(2)=f(1,1)>1$, then by monotonicity, we get $v(3)=f(1,v(2))\geq f(1,1)=v(2)$, and, in general, $v(n+1)\geq v(n)$. Thus, in this case, the sequence $v(n)$ is (non-strictly) increasing.

Similarly, if $v(2)=f(1,1)<1$, then we get $v(3)\leq v(2)$ and, in general, $v(n+1)\leq v(n)$, i.e., in this case, the sequence $v(n)$ is strictly decreasing.

Let us consider these two cases one by one.

3.2.1°. Let us first consider the case when the sequence $v(n)$ is increasing. In this case, for every three integers m, n, and p, if $2^m\leq p^n$, then $v(2^m)\leq v(p^n)$, i.e., $(v(2))^m\leq(v(p))^n$.

For all m, n, and p, the inequality $2^m \le p^n$ is equivalent to $m \cdot \ln(2) \le n \cdot \ln(p)$, i.e., to $\dfrac{m}{n} \le \dfrac{\ln(p)}{\ln(2)}$. Similarly, the inequality $(v(2))^m \ge (v(p))^n$ is equivalent to $\dfrac{m}{n} \le \dfrac{\ln(v(p))}{\ln(v(2))}$. Thus, the above conclusion "if $2^m \le p^n$ then $(v(2))^m \le (v(p))^n$" takes the following form:

$$\text{for every rational number } \frac{m}{n}, \text{ if } \frac{m}{n} \le \frac{\ln(p)}{\ln(2)} \text{ then } \frac{m}{n} \le \frac{\ln(v(p))}{\ln(v(2))}.$$

Similarly, for all m', n', and p, if $p^{n'} \le 2^{m'}$, then $v(p^{n'}) \le v(2^{m'})$, i.e., $(v(p))^{n'} \le (v(2))^{m'}$. The inequality $p^{n'} \le 2^{m'}$ is equivalent to $n' \cdot \ln(p) \le m' \cdot \ln(2)$, i.e., to $\dfrac{\ln(p)}{\ln(2)} \le \dfrac{m'}{n'}$. Also, the inequality $(v(p))^{n'} \le (v(2))^{m'}$ is equivalent to $\dfrac{\ln(v(p))}{\ln(v(2))} \le \dfrac{m'}{n'}$. Thus, the conclusion "if $p^{n'} \le 2^{m'}$ then $(v(p))^{n'} \le (v(2))^{m'}$" takes the following form:

$$\text{for every rational number } \frac{m'}{n'}, \text{ if } \frac{\ln(p)}{\ln(2)} \le \frac{m'}{n'} \text{ then } \frac{\ln(v(p))}{\ln(v(2))} \le \frac{m'}{n'}.$$

Let us denote $\gamma \stackrel{\text{def}}{=} \dfrac{\ln(p)}{\ln(2)}$ and $\beta \stackrel{\text{def}}{=} \dfrac{\ln(v(p))}{\ln(v(2))}$. For every $\varepsilon > 0$, there exist rational numbers $\dfrac{m}{n}$ and $\dfrac{m'}{n'}$ for which $\gamma - \varepsilon \le \dfrac{m}{n} \le \gamma \le \dfrac{m'}{n'} \le \gamma + \varepsilon$. For these numbers, the above two properties imply that $\dfrac{m}{n} \le \beta$ and $\beta \le \dfrac{m'}{n'}$ and thus, that $\gamma - \varepsilon \le \beta \le \gamma + \varepsilon$, i.e., that $|\gamma - \beta| \le \varepsilon$. This is true for all $\varepsilon > 0$, so we conclude that $\beta = \gamma$, i.e., that $\dfrac{\ln(v(p))}{\ln(v(2))} = \gamma$. Hence, $\ln(v(p)) = \gamma \cdot \ln(p)$ and thus, $v(p) = p^\gamma$ for all integers p.

3.2.2°. We can reach a similar conclusion $v(p) = p^\gamma$ when the sequence $v(n)$ is decreasing and $v(2) < 1$, and a conclusion that $v(p) = 0$ if $v(2) = 0$.

3.3°. By definition of $v(n)$, we have $f(v(m), v(m')) = v(m + m')$. Thus, we have $f(m^\gamma, (m')^\gamma) = (m + m')^\gamma$. By using scale-invariance with $\lambda = n^{-\gamma}$, we get $f\left(\dfrac{m^\gamma}{n^\gamma}, \dfrac{(m')^\gamma}{n^\gamma}\right) = \dfrac{(m + m')^\gamma}{n^\gamma}$. Thus, for $a = \dfrac{m^\gamma}{n^\gamma}$ and $b = \dfrac{(m')^\gamma}{n^\gamma}$, we get $f(a, b) = (a^\alpha + b^\alpha)^{1/\alpha}$, where $\alpha \stackrel{\text{def}}{=} 1/\gamma$.

Rational numbers $r = \dfrac{m}{n}$ are everywhere dense on the real line, hence the values r^γ are also everywhere dense, i.e., every real number can be approximated, with any given accuracy, by such numbers. Thus, continuity implies that $f(a, b) = (a^\alpha + b^\alpha)^{1/\alpha}$ for every two real numbers a and b.

The proposition is proven.

Acknowledgments

This work was supported in part by the National Science Foundation via grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science) and HRD-1242122 (Cyber-ShARE Center of Excellence).

References

[1] J. Aczel and J. Dhombres, *Functional Equations in Several Variables*, Cambridge University Press, New York, 2008.

[2] K. Autchariyapanitkul, O. Kosheleva, V. Kreinovich, and S. Sriboonchitta, "Quantum econometrics: how to explain its quantitative successes and how the resulting formulas are related to scale invariance, entropy, and fuzziness", In: V.-N. Huynh, M. Inuiguchi, D.-H. Tran, and Th. Denoeux (eds.), *Proceedings of the International Symposium on Integrated Uncertainty in Knowledge Modelling and Decision Making IUKM'2018*, Hanoi, Vietnam, March 13–15, 2018.

[3] 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.

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

[5] A. Farhan, O. Kosheleva, and V. Kreinovich, "Why max and average poolings are optimal in convolutional neural networks", *Proceedings of the 7th International Symposium on Integrated Uncertainty in Knowledge Modelling and Decision Making IUKM'2019*, Nara, Japan, March 27–29, 2019.

[6] O. Fuentes, J. Parra, E. Anthony, and V. Kreinovich, "Why rectified linear neurons are efficient: a possible theoretical explanations", In: O. Kosheleva, S. Shary, G. Xiang, and R. Zapatrin (eds.), *Beyond Traditional Probabilistic Data Processing Techniques: Interval, Fuzzy, etc. Methods and Their Applications*, Springer, Cham, Switzerland, 2019, to appear.

[7] 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.

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

[9] P. C. Kainen, V. Kurkova, V. Kreinovich, and O. Sirisaengtaksin. "Uniqueness of network parameterization and faster learning", *Neural, Parallel, and Scientific Computations*, 1994, Vol. 2, pp. 459–466.

[10] 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.

[11] V. Kreinovich, "Group-theoretic approach to intractable problems," *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Vol. 417, 1990, pp. 112–121.

[12] 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.) *Proceedings of the World Conference on Soft Computing*, Baku, Azerbaijan, May 29–31, 2018.

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

[14] G. Muela, C. Servin, and V. Kreinovich, "How to make machine learning robust against adversarial inputs", *Mathematical Structures and Modeling*, 2017, Vol. 42, pp. 127–130.

[15] H. T. Nguyen and V. Kreinovich, *Applications of Continuous Mathematics to Computer Science*, Kluwer, Dordrecht, Netherlands, 1997.

[16] J. Parra, O. Fuentes, E. Anthony, and V. Kreinovich, "Prediction of volcanic eruptions: case study of rare events in chaotic systems with delay", *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics SMC'2017*, Banff, Canada, October 5–8, 2017, pp. 351–356.

[17] J. Parra, O. Fuentes, E. Anthony, and V. Kreinovich, "Use of machine learning to analyze and – hopefully – predict volcano activity", *Acta Politechnica Hungarica*, 2017, Vol. 14, No. 3, pp. 209–221.

[18] 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, GA, May 28–31, 1995, Vol. 1, pp. 419–423.

[19] N. Wiener, *Cybernetics: Or Control and Communication in the Animal and the Machine*, MIT Press, Cambridge, Massachisetts, 1948.