# Why Rectified Linear Neurons Are Efficient: A Possible Theoretical Explanation

Olac Fuentes, Justin Parra, Elizabeth Anthony, and Vladik Kreinovich

**Abstract** Traditionally, neural networks used a sigmoid activation function. Recently, it turned out that piecewise linear activation functions are much more efficient – especially in deep learning applications. However, so far, there have been no convincing theoretical explanation for this empirical efficiency. In this paper, we provide such an explanation.

## 1 Rectified Linear Neurons: Formulation of the Problem

**Why neural networks: a brief reminder.** One of the main objectives of designing computers is that they would solve *intelligent* tasks, tasks that we normally solve by using our brains. It is therefore reasonable, when designing computational devices, to emulate how our brain works.

In the brain, signals come from the special sensor cells in the eyes, ears, etc., and are processed by other cells called *neurons*. The signals from the sensors come as series of electric spikes. The intensity of the corresponding signal is reflected by the frequency of the spikes.

Signal processing cells – neurons – usually:

- take inputs from several cells (sensor cells or other data processing neurons),
- process the summary input signal, and
- send the resulting signal to other neurons – or to the cells that perform some activities (e.g., move a finger, close an eye, slow down the heart rate, etc.).

Olac Fuentes, Justin Parra, and Vladik Kreinovich
Department of Computer Science, University of Texas at El Paso, El Paso, TX 79968, USA
e-mail: ofuentes@utep.edu, jrparra2@miners.utep.edu, vladik@utep.edu

Elizabeth Anthony
Department of Geological Sciences, University of Texas at El Paso, El Paso, Texas 79968, USA
e-mail: eanthony@utep.edu

To be more precise, when a neuron gets signals $x_1, \ldots, x_n$ from different inputs:

- these signals are first aggregated into a linear combination

$$x = w_1 \cdot x_1 + \ldots + w_n \cdot x_n + w_0,$$

  and then
- an appropriate transformation $y = s_0(x)$ is applied to the aggregated signal $x$.

As a result, we get the output

$$y = s_0(w_1 \cdot x_1 + \ldots + w_n \cdot x_n + w_0). \tag{1}$$

The corresponding function $s_0(x)$ is known as the *activation function*; see, e.g., [2].

This is exactly how the standard artificial neural networks – that emulate biological neural networks – work:

- we feed the inputs $x_i$ into one or more neurons, then
- we feed these neuron's outputs into other neurons, etc.

We can have simple networks, in which inputs go into the intermediate layer, and the outputs of the intermediate layer are collected by neurons from the final layer. We can have neural networks with more layers. Interestingly, it turns out that *deep learning* neural networks – i.e., networks with a large number of layers – are the most efficient ones; see, e.g., [3].

**Which activation functions are most effective.** In the past, most neural networks used the *sigmoid* activation functions $s_0(x) = \dfrac{1}{1+\exp(-k \cdot x)}$, the activation function which provides the most adequate description of data processing in biological neurons.

However, recently, it was shows that we can make neural networks more efficient if instead, we use *rectified linear* neurons, with piecewise linear activation function $s_0(x) = \max(x, 0)$, i.e.:

- $s_0(x) = x$ when $x \geq 0$, and
- $s_0(x) = 0$ for $x < 0$.

Such neurons are especially efficient in *deep learning* [3].

In particular, we successfully used rectified linear neurons to predict volcanic eruptions based on preceding seismic activity; see, e.g., [9, 10].

*Comment.* It is easy to prove that 3-layer neural networks with rectified linear neurons are universal approximators for continuous functions on a bounded domain. Indeed:

- each function can be represented as a difference of two convex functions (see, e.g., [13]), and
- each convex function is a maximum of all tangent linear functions – and thus, can be well approximate if we take finitely many tangent linear functions [13].

**Why are rectified linear neurons efficient: an open question.** While empirical evidence shows that rectified linear neurons work best, there seems to be no convincing theoretical explanation for this empirical success. Without such an explanation, it is not clear whether these neurons are indeed the best – or maybe some other activation function would lead to even more efficient computations?

**What we do in this paper.** In this paper, we provide a theoretical explanation of why rectified linear activation functions are empirically successful.

## 2 Our Explanation

**What do we mean by optimal?** We are interested in finding *optimal* activation functions, i.e., functions which are the best according to some 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 an *order* $\preceq$ 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 $a$.

In our case, we want to select the best activation function. Thus, by an optimality criterion, we would mean an order on the set of all possible objective functions.

In these terms, a function $s_0(x)$ is optimal if is better (or of the same quality) than all other possible activation functions, i.e., if $s \preceq s_0$ for all possible activation functions $s(x)$.

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

What if several different functions 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 activation functions, we, in effect, replace it with a modified criterion, and keep modifying it until *finally* we get a criterion for which only one activation function is optimal. So, we arrive at the following definition.

**Definition 1.**

- *By an* optimality criterion*, we mean a (partial) order $\preceq$ on the set of all continuous functions of one variable.*
- *We say that a function $s_0$ is* optimal *with respect to the optimality criterion $\preceq$ if $s \preceq s_0$ for all functions $s$.*
- *We say that an optimality criterion is* final *if there exists exactly one function which is optimal with respect to this optimality criterion.*

**Numerical values depend on the measuring unit.** Which optimality criterion should we use? In selecting the optimality criterion, we should take into account that when we measure a physical signal, the resulting numerical value depends on what measuring unit we use in this measurement. For example, 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 $\lambda$ times smaller than the previous one, then all the numerical values get multiplied by $\lambda$; e.g., if we replace meters by centimeters, all numerical values get multiplied by $\lambda = 100$.

**This is important for neural networks, even though inputs are usually normalized.** 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.

**Scale-invariance.** 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 requirement in precise terms.

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 $\lambda$ 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 2.** *We say that an optimality criterion $\preceq$ is* scale-invariant *if for every two functions $s$ and $s'$ and for every $\lambda > 0$, the relation $s \preceq s'$ is equivalent to $T_\lambda(s) \preceq T_\lambda(s')$, where we denoted $(T_\lambda(s))(x) \stackrel{\text{def}}{=} \lambda^{-1} \cdot s(x)$.*

Now, we are ready to formulate our result.

**Proposition 1.** *A function $s_0(x)$ is optimal with respect to some final scale-invariant optimality criterion if and only if 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 1.* 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.*

*Comment 2.* 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 scale-invariant.

**Proof of Proposition 1.**

1°. For every function $s_0(x)$ of the above type, we can easily find a final scale-invariant optimality criterion for which this function is optimal: namely, we can take the order $\preceq$ in which $s \preceq s_0$ for all continuous functions $s(x)$.

One can easily check:

- that this relation is final and scale-invariant, and
- that the given function $s_0(x)$ is the only function which is optimal with respect to this criterion.

2°. Vice versa, let us assume that a function $s_0(x)$ is optimal with respect to some final scale-invariant optimality criterion. Under this assumption, we need to prove that the function $s_0(x)$ has the desired form. To prove this, let us prove that this function is scale-invariant in the sense of Definition 1.

In terms of the transformation $T_\lambda$, scale-invariance means that $s_0 = T_\lambda(s_0)$ for all $s$. To prove that $T_\lambda(s_0) = s_0$, let us prove that the function $T_\lambda(s_0)$ is optimal. Then, the desired equality will follow from the fact that the optimality criterion is final – and thus, there is only one optimal function.

To prove that the function $T_\lambda(s_0)$ is optimal, we need to prove that $s \preceq T_\lambda(s_0)$ for all $s$. Due to scale-invariance of the optimality criterion, this condition is equivalent to $T_{\lambda^{-1}}(s) \preceq s_0$ – which is, of course, always true, since $s_0$ is optimal. Thus, $T_\lambda(s_0)$ is also optimal, hence $T_\lambda(s_0) = s_0$ for all $\lambda$.

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 conclusion.

3°. Every input $x$ is either equal to 0, or positive, or negative. Let us consider these three cases one by one.

4°. 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 $y = s_0(0) = 0$.

5°. Let us now consider the case of positive values $x$.

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

- $x$ instead of $\lambda$,

- 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.

$6^\circ$. 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.

## 3 Auxiliary Arguments in Favor of Rectified Linear Neurons

We have proved that for every reasonably optimality criterion, the optimal activation function corresponds to rectified linear neurons. To make this mathematical result more intuitively convincing, let us provide some informal arguments explaining the advantages of such activation functions.

### *3.1 Symmetry-Based Argument*

**Numerical values depend on the measuring unit.** As we have mentioned in the previous section, when we measure a physical signal, the resulting numerical value depends on what measuring unit we use in this measurement. The choice of a measuring unit is rather arbitrary, it does not change the physical situation. It is reasonable to require that the results of applying the corresponding non-linear activation function not change is we simply change the measuring unit.

In precise terms, this means that if we have $y = s_0(x)$, then for any $\lambda > 0$, we should have $y' = s_0(x')$, where we denoted $x' = \lambda \cdot x$ and $y' = \lambda \cdot y$. Let us see what we can derive based on this requirement.

**Definition 3.** *We say that a function $s_0(x)$ is a* scale-invariant *if, for every $x$, $y$, and $\lambda > 0$, $y = s_0(x)$ implies that $\lambda \cdot y = s_0(\lambda \cdot x)$.*

**Proposition 2.** *A function $s_0(x)$ is scale-invariant if and only if it has the following form:*

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

*for some constants* $c_+$ *and* $c_-$.

**Proof:** this result was, in effect, proven when we proved Proposition 1 – see Parts 3-6 of this proof.

*Comment 1.* It should be mentioned that it is well known – and very easy to check – that the activation function corresponding to rectified linear neurons is scale-invariant. What we prove is slightly more complex: namely, we also show that rectified linear functions are the *only* scale-invariant activation functions.

*Comment 2.* It is also important to emphasize that neither this informal argument (nor two other arguments that we present next) replace the formal proof. Their only purpose is to make the result of the above mathematical proof more intuitive and thus, more convincing.

## *3.2 Complexity-Based Argument*

**Idea.** To speed up computations, we need to make sure that the activation function is as fast to compute as possible.

**This idea leads to another intuitive argument in favor of rectified linear neurons.** Inside the computer, every numerical operation is implemented as a composition of the basic hardware-supported operations. These operations include the basic arithmetic operations:

- addition $a+b$,
- subtraction $a-b$,
- multiplication $a \cdot b$,
- division $a/b$,

and the operations $\min(a,b)$ and $\max(a,b)$.

Of these operations:

- the functions min and max are the fastest,
- addition $+$ and subtraction $-$ are next fastest,
- followed by multiplication (which involves several additions) and
- division (which involves several multiplications);

see, e.g., [11].

The fastest-to-compute activation function is the one that uses only one hardware supported basic operation.

We are interested in non-linear activation functions (since linear transformation are already taken care in the aggregation procedure, before we invoke the activation function). Out of the above operations, the corresponding functions $s_0(a) = a + a_0$, $s_0(a) = a - a_0$, $s_0(a) = a_0 - a$, $s_0(a) = a \cdot a_0$, and $s_0(a) = a/a_0$ are linear. The only

non-linear operations are $\max(a,a_0)$, $\min(a,a_0)$, and $a_0/a$. Of these three operations, the fastest are piecewise linear operations min and max.

Thus, the computational complexity-based analysis indeed leads to yet another argument in favor of piecewise linear activation functions.

*Comment 1.* This complexity-based argument is very simple and straightforward. We want to once again emphasize that the fact that rectified linear activation functions are fast-to-compute does not entail that will lead to accurate learning. However, this fact does – at least in our opinion – make our theoretical result somewhat more intuitively convincing.

*Comment 2.* A similar argument can be made if we are thinking about a hardware implementation of artificial neural networks. Indeed, in this case, a linear combination is straightforward: just place several currents together.

The simplest nonlinear element of an electric circuit is a *diode* that transmits current only in one direction. For the diode, the output is equal to $x$ if $x \geq 0$ and to 0 otherwise, i.e., it is exactly the rectified linear activation function – which is thus the easiest to implement in hardware.

## *3.3 Fuzzy-Based Argument*

**Need to use fuzzy techniques.** When we use neural network technique to learn a phenomenon, we generate a neural network that provides a good approximation to this phenomenon. In particular, when we use the neural network technique to provide a solution to a problem – e.g., to provide an appropriate control – we thus produce a neural network that generates the corresponding solution.

In human reasoning, we try our best not only to provide good solutions to real-life problems, but also to provide a clear justification for these solutions.

It is therefore reasonable to look for activation functions for which the corresponding solution makes direct sense, i.e., for which this solution can be interpretable in human-understandable natural-language terms.

The need for translating imprecise ("fuzzy") expert knowledge into precise (and thus, computer-understandable) form has been well recognized since the early 1960s. Techniques that provide such a translation are known as *fuzzy techniques*; see, e.g., [1, 4, 6, 7, 14].

In terms of these techniques, the above idea can be reformulated as follows: we want to select an activation function for which all the functions representing the corresponding neural networks are directly interpretable in fuzzy terms.

**Which functions can be interpretable in fuzzy terms.** It is known that if we use $1-a$ as negation, $\min(a+b,1)$ as an "or"-operation and $\max(a+b-1,0)$ as an "and"-operation, then functions that can be represented as compositions of logical operations are exactly piece-wise linear functions with integer coefficients [5, 8, 12].

To these operations, we can add more subtle operations. For example, it is natural to interpret "somewhat $A$" as $A \vee A$ – which, in the above logic, leads to $2a$ (or, to

be more precise, to $\min(2a, 1)$). It is therefore reasonable to define an inverse hedge "very $A$" as the statement $B$ for which "somewhat $B$" is equivalent to $A$. In the above logic, this would mean defining our degree of confidence in "very $A$" as $a/2$, where $A$ is our degree of confidence in the original statement $A$.

We can iterate this "very" hedge, thus getting values $a/4$, $a/8$, etc. By combining these hedges and logical operations, we can get any piecewise linear functions with binary-rational coefficients.

**This leads to a new argument in favor of piecewise linear activation functions.** We want a neural network to be interpretable. For the neural network to be interpretable, we need to make sure that all the data processing algorithms performed by a neural network can be described in fuzzy terms. Since implies that all such algorithms must be piecewise-linear.

This conclusion means, in particular, that the activation function should be piecewise linear. Thus, we indeed get one more argument in favor of using piecewise linear activation functions in neural networks.

*Comment.* Similarly to the previous two arguments, this argument is not, by itself, a substitute for the proof: the results of neural network training are usually not easy to understand and interpret anyway. However, as with the previous two arguments, this argument hopefully make our formal proof somewhat more intuitively convincing.

## Acknowledgments

This work was supported in part by the US National Science Foundation grant HRD-1242122.

## References

1. R. Belohlavek, J. W. Dauben, and G. J. Klir, *Fuzzy Logic and Mathematics: A Historical Perspective*, Oxford University Press, New York, 2017.
2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2006.
3. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Leaning*, MIT Press, Cambridge, Massachusetts, 2016.
4. G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.
5. R. McNaughton, "A theorem about infinite-valued sentential logic", *Journal of Symbolic Logic*, 1951, Vol. 16, pp. 1–13.
6. J. M. Mendel, *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions*, Springer, Cham, Switzerland, 2017.
7. H. T. Nguyen and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2006.
8. V. Novák, I. Perfilieva, and J.. Močkoř, *Mathematical Principles of Fuzzy Logic*, Kluwer, Boston, Dordrecht, 1999.

9. 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.
10. J. Parra, O. Fuentes, E. Anthony, and V. Kreinovich, "Use of machine learning to analyze and – hopefully – predict volcano activity", *Acta Politechnica Hungarica*, to appear.
11. D. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufman, Waltham, Massachusetts, 2012.
12. I. Perfilieva and A. Tonis, "Functional system in fuzzy logic formal theory", *BUSEFAL*, 1995, Vol. 64, pp. 42–50.
13. R. T. Rockafeller, *Convex Analysis*, Princeton University Press, Princeton, New Jersey, 1970.
14. L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.