2-2020

Why Squashing Functions in Multi-Layer Neural Networks

Julio Urenda
The University of Texas at El Paso, jcurenda@utep.edu

Orsoly Csiszár
Óbuda University, orsolya.csiszar@nik.uni-obuda.hu

Gábor Csiszár
University of Stuttgart, gabor.csiszar@mp.imw.uni-stuttgart.de

József Dombi
University of Szeged, dombi@inf.u-szeged.hu

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

See next page for additional authors

Comments:

Technical Report: UTEP-CS-20-12

Recommended Citation

Urenda, Julio; Csiszár, Orsoly; Csiszár, Gábor; Dombi, József; Kosheleva, Olga; Kreinovich, Vladik; and Eigner, György, "Why Squashing Functions in Multi-Layer Neural Networks" (2020). *Departmental Technical Reports (CS)*. 1398.
https://scholarworks.utep.edu/cs_techrep/1398

Why Squashing Functions in Multi-Layer Neural Networks

1st Julio C. Urenda
Department of Mathematical Sciences
and Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
jcurenda@utep.edu

2nd Orsolya Csiszár
Faculty of Basic Sciences
University of Applied Sciences Esslingen
Esslingen, Germany, and
Institute of Applied Mathematics
Óbuda University
Budapest, Hungary
orsolya.csiszar@nik.uni-obuda.hu

3rd Gábor Csiszár
Institute of Materials Physics
University of Stuttgart
Stuttgart, Germany
gabor.csiszar@mp.imw.uni-stuttgart.de

4th József Dombi
Institute of Informatics
University of Szeged
Szeged, Hungary
dombi@inf.u-szeged.hu

5th Olga Kosheleva
Department of Teacher Education
University of Texas at El Paso
El Paso, TX 79968, USA
olgak@utep.edu

6th Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
vladik@utep.edu

7th György Eigner
Institute of Applied Mathematics
Óbuda University
Budapest, Hungary
eigner.gyorgy@nik.uni-obuda.hu

***Abstract*—Most multi-layer neural networks used in deep learning utilize rectified linear neurons. In our previous papers, we showed that if we want to use the exact same activation function for all the neurons, then the rectified linear function is indeed a reasonable choice. However, preliminary analysis shows that for some applications, it is more advantageous to use different activation functions for different neurons – i.e., select a family of activation functions instead, and select the parameters of activation functions of different neurons during training. Specifically, this was shown for a special family of *squashing* functions that contain rectified linear neurons as a particular case. In this paper, we explain the empirical success of squashing functions by showing that the formulas describing this family follow from natural symmetry requirements.**

***Index Terms*—multi-layer neural networks, rectified linear function, squashing function, invariance**

I. Formulation of the Problem

Machine learning is needed to analyze systems of systems. For some simple systems, we know the equations that describe the system's dynamics. These equations may be approximate, but they are often good enough. With more complex systems (such as systems of systems), this is often no longer the case. Even when we have a good approximate model for each subsystem, the corresponding inaccuracies add up, and the resulting model of the whole system is too inaccurate to be useful. For real-life systems like a city or a big plant, it is therefore often not possible to predict the system's behavior based only on approximate models of subsystems. We also need to use the records of the actual system's behavior when making such predictions. Techniques that use the previous behavior of a system to predict its future behavior are known as *machine learning* techniques.

This work was supported in part by the grant TUDFO/47138-1/2019-ITM from the Ministry of Technology and Innovation, Hungary, and by the US National Science Foundation 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). The authors are greatly thankful to Professor Imre Rudas for his encouragement.

Deep learning. At present, the most efficient machine learning technique is *deep learning*, i.e., the use of multi-layer neural networks; see, e.g., [6]. In general, on layer of a neural network, we transform signals $x_1, \ldots, x_n$ into a new signal $y = s\left(\sum\limits_{i=1}^{n} w_i \cdot x_i + w_0\right)$, where the coefficient w_i (called *weights*) are to be determined during training, and $s(z)$ is a non-linear function called *activation function*.

Activation functions used in deep learning. Most multi-layer neural networks used in deep learning utilize rectified linear neurons, i.e., neurons that use the activation function $s(z) = \max(z, 0)$ known as *rectified linear* function.

Why rectified linear activation function. In our previous papers [5], [7], we use invariance ideas – similar to what we will use later in this paper – to show that if we want to use the exact same activation function for all the neurons, then the rectified linear function is indeed a reasonable choice.

Shall we go beyond rectified linear activation functions? Preliminary analysis shows that for some applications, it is more advantageous to use different activation functions for different neurons – i.e., select a family of activation functions instead, and select the parameters of activation functions of different neurons during training. Specifically, this was shown for a special family of *squashing* activation functions that contain rectified linear neurons as a particular case; see, e.g., [2]–[4]. Functions from this family have the form

$$S^{(\beta)}_{a,\lambda}(z) = \frac{1}{\lambda \cdot \beta} \cdot \ln \frac{1 + \exp(\beta \cdot z - (a - \lambda/2))}{1 + \exp(\beta \cdot z - (a + \lambda/2))}. \quad (1)$$

Why squashing functions? In this paper, we explain the empirical success of squashing functions by showing that the formulas describing this family also follow from natural symmetry requirements.

How this paper is structured. In Section 2, we recall the main ideas of symmetries and invariance. In Section 3, we recall how these ideas can be used to explain the efficiency of the sigmoid activation function

$$s(z) = \frac{1}{1 + \exp(-z)} \quad (2)$$

in the traditional 3-layer neural networks. Finally, in Section 4, we use this information to explain the efficiency of squashing activation functions.

II. Natural Symmetries: General Idea

Numerical values change when we change a measuring unit and/or starting point. In data processing, we deal with numerical values of different physical quantities. Computers just treat these values as numbers, but from the physical viewpoint, it is important to understand that the numerical values are not absolute: they change if we change the measuring unit and/or the starting point for measuring the corresponding quantity.

For example, we can measure a person's height in meters or in centimeters. The same height of 1.7 m, when described in centimeters, becomes 170 cm. In general, if we replace the original measuring unit with a new unit which is λ times smaller, then for each physical quantity, instead of the original numerical value x, we get a new numerical value $\lambda \cdot x$ – while the actual quantity remains the same.

For some physical quantities, e.g., for time or temperature, the numerical value also depends on the starting point. For example, we can measure the time by counting how much time has passed during the flight – i.e., by using the flight start time as a starting point. Alternatively, we can use the usual calendar time, in which Year 0 is the starting point. In general, if we replace the original starting point with the new one which is x_0 units earlier, than each original numerical value x is replaced by a new numerical value $x + x_0$.

In general, if we change both the measuring unit and the starting point, we get a linear transformation: from the original value x, we get to $\lambda \cdot x + x_0$. A usual example of such a transformation is a transition from Celsius to Fahrenheit temperature scales: if we know the temperature t_C is Celsius, then the Fahrenheit temperature t_F is equal to $t_F = 1.8 \cdot t_C + 32$.

Invariance. Changing the measuring unit and/or starting point changes the numerical values but does not change the actual quantity. It is therefore reasonable to require that physical equations do not change if we simply change the measuring unit and/or change the starting point.

Of course, to preserve the physical equations, if we change the measuring unit and/or starting point for one quantity, we may need to change the measuring units and/or starting points for other quantities as well. For example, there is a well-known relation $d = v \cdot t$ between distance d, velocity v, and time t. If we change the measuring units for measuring distance and time, this formula remains valid – but only if we accordingly change the units for velocity. For example, if we replace kilometers with meters and hours with seconds, then, to preserve this formula, we also need to change the unit for velocity from km/h to m/sec.

Natural transformations beyond linear ones: analysis. In the previous text, we considered only linear transformations between different scales. In some cases, the relation between different scales is non-linear. For example, we can measure the earthquake energy is Joules (i.e., in the usual scale) or in a logarithmic (Richter) scale.

The possibility of non-linear transformations raises a natural question: what are the natural transformations between different scales?

- First, as we have argued in the previous text, all linear transformations are natural.
- Second, if we have a natural transformation $f(x)$ from scale A to another B, then the inverse transformation $f^{-1}(x)$ from scale B to scale A should also be natural.
- Third, if $f(x)$ and $g(x)$ are natural scale transformation, then we can apply first $g(x)$ to get $y = g(x)$ and then f to get $f(y) = f(g(x))$. Thus, the composition $f(g(x))$ of two natural transformations should also be natural.

In mathematical terms, the class of transformations that contain an identity mapping $f(x) = x$ and that satisfies the second and third properties is called a *transformation group*. In these terms, the above properties can be reformulated as follows: the class T of natural transformations is a transformation group that contains all linear transformations.

We also need to take into account that in a computer, at any given moment of time, we can only store the values of finitely many parameters. Thus, the transformations from the desired transformation group T should be determined by a finite number of parameters. In mathematical terms, the smallest number of parameters needed to describe a family is known as the *dimension* of this family – just like the fact that we need 3 coordinates to describe any point in space means that the physical space is 3-dimensional. In these terms, the transformation group T must be finite-dimensional.

Let us describe all natural transformations. Interestingly, the above requirements uniquely determine the class of all possible natural transformation. This result can be traced back to Norbert Wiener, the father of cybernetics. In his seminal book *Cybernetics* [10] that started this research area, he noticed that when we approach an object form afar, our perception of this object goes through several distinct phases:

- first, we see a blob; this means that at a large distance, we cannot distinguish between images obtained each other by all possible continuous transformations; in other words, this phase corresponds to the group of all possible continuous transformations; transformations);

- as we get closer, we start distinguishing angular parts from smooth parts, but still cannot compare sizes; this corresponds to the group of all projective transformations;
- after that, we become able to detect parallel lines; this corresponds to the group of all transformations that preserve parallel lines – i.e., to the group of all linear (= affine) transformations;
- when we get even closer, we become able to detect the shapes, but we still cannot distinguish between larger objects that are further away and smaller objects which are closer – our binocular vision (that enables us to make this distinction) only starts working at shorter distances; this corresponds to the group of all homotheties;
- finally, as we get much closer, we see the exact shapes and sizes; this means that only the identity transformation remains.

Wiener argued that there are no other transformation groups – since if there were other transformation groups, after billions years of evolutions, we would use them. In precise terms, he conjectured that the only two finite-dimensional transformation groups that contain all linear transformations are the groups of all linear transformations and the group of all projective transformations. For transformations of the real line, projective transformations are simply fractional-linear transformations; see, e.g., [8], [9] and references therein:

$$f(x) = \frac{a\cdot x+b}{c\cdot x+d}.$$

III. How Invariance Ideas Explain the Efficiency of Sigmoid Activation Functions in Traditional Neural Networks

Why traditional neural networks. In order to understand why sigmoid functions are efficient neural networks, let us recall why traditional neural networks appeared in the first place; see, e.g., [7].

The main reason, in our opinion, was that computers were too slow. A natural way to speed up computations is to make several processors work in parallel – so that each processor only handles a simple task, not requiring too much computation time.

For processing data, the simplest possible functions to compute are linear functions. However, we cannot only use linear functions – because then, no matter how many linear transformations we apply one after another, we will only get linear functions, and many real-life dependencies are nonlinear. So, we need to supplement linear computations with some nonlinear ones. In general, the fewer inputs, the faster the computations. Thus, the fastest to compute are functions with one input, i.e., functions of one variable. So, we end up with a parallel computational device that has linear processing units (L) and nonlinear processing units (NL) that compute functions of one variable. First, the input signals come to a layer of such devices, then the results of this layer go to another layer, etc. The fewer layers we have, the faster the computations.

It can be shown (see, e.g., [7]) that 1-layer schemes (L or NL) and 2-layer schemes (L-NL, linear layer followed by non-linear layer, or NL-L) are not sufficient to approximate any possible dependence. Thus, we need at least 3-layer networks – and 3-layer networks can be proven to be sufficient. In a 3-layer network, we cannot have two linear layers or two nonlinear layers following each other – that would be equivalent to having one layer since, e.g., a composition of two linear functions is also linear. Thus, we have only two options: L-NL-L and NL-L-NL. Since linear transformations are faster to compute, the fastest scheme is L-NL-L. In this scheme:

- first, each neuron k in the L layer combines the inputs into a linear combination $z_k = \sum\limits_{i=1}^{n} w_{ki}\cdot x_i + w_{k0}$;
- then, in the next layer, each such signal is transformed into $y_k = s_k(z_k)$ for some non-linear function; and
- finally, in the last linear layer, we form a linear combination of the values y_k: $y = \sum\limits_{k=1}^{K} W_k\cdot y_k + W_0$.

The resulting transformation takes the form

$$y = \sum_{k=1}^{K} W_k\cdot s_k\left(\sum_{i=1}^{n} w_{ki}\cdot x_i + w_{k0}\right) + W_0.$$

Usually, we use the same function $s(z)$ for all transformations, so we get

$$y = \sum_{k=1}^{K} W_k\cdot s\left(\sum_{i=1}^{n} w_{ki}\cdot x_i + w_{k0}\right) + W_0.$$

This is indeed the usual formula of the traditional neural network.

Why sigmoid activation function: an idea behind the invariance-based explanation. In real life, signals come with noise, in particular, with background noise that, in effect, adds a constant to all the measured signals. We can try to get rid of this noise by subtracting the corresponding constant, i.e., by replacing the original numerical values x_i with a corrected value $x_i - n_i$. After this correction, instead of the original value z_k, we get a corrected value

$$z'_k = \sum_{i=1}^{n} w_{ki}\cdot (x_i - n_i) + w_{k0} = z_k - h'_k,$$

where we denoted $h'_k \overset{\text{def}}{=} \sum\limits_{i=1}^{n} w_{ki}\cdot n_i$.

The trouble is that we do not know the exact value of this constant – otherwise, this noise would not be a problem. So, depending on our estimate, we may subtract different values n_i and thus, different values h'_k. If we change from one value h'_k to another one h''_k, then the resulting value of z_k is shifted by the difference $h_k \overset{\text{def}}{=} h'_k - h''_k$, namely, $z''_k = z'_k + h_k$, exactly the same formula as for the shift corresponding to the change in the starting point.

Since we do not know what shift is the best, all shifts within a certain range are equally possible. It is therefore reasonable to require that the formula $y = s(z)$ for the nonlinear activation function should work for all possible shifts. In other words, as we mentioned in the previous section, if we shift from z to $z' = z + h$, then we should satisfy the exact same formula $y' = s(z')$ – probably for an appropriately transformed value y.

In the previous section, we also mentioned that all possible transformations should be fractionally linear. Thus, for each possible shift h, the value $s(z') = s(z+h)$ should be obtained from $s(z)$ by an appropriate fractionally linear transformation:

$$s(z+h) = \frac{a(h)\cdot s(z) + b(h)}{c(h)\cdot s(z) + d(h)}. \qquad (3)$$

Let us show that this implies the sigmoid.

Why sigmoid – derivation: generic case. For $h = 0$, we should have $s(z+h) = s(z)$, thus, we should have $d(0) \neq 0$. It is reasonable to require that the function $d(h)$ is continuous. In this case, $d(h)$ is different from 0 for all small h. Then, we can divide both numerator and denominator of the formula (3) by $d(h)$ and get a simpler formula, with only three functions of h:

$$s(z+h) = \frac{A(h)\cdot s(z) + B(h)}{C(h)\cdot s(z) + 1}, \qquad (4)$$

where we denoted $A(h) = a(h)/d(h)$, $B(h) = b(h)/d(h)$, and $C(h) = c(h)/d(h)$. For $h = 0$, we have $s(z+h) = s(z)$, so $A(h) = 1$ and $B(h) = C(h) = 0$.

It is also reasonable to require that the activation function $s(z)$ be smooth. We also want it to be defined for all z.

Smoothness requirement comes from the fact that on each interval, every continuous function can be approximated, with any desired accuracy, by a smooth one – even by a polynomial. So we can always get non-smooth functions as limits of smooth ones.

Multiplying both sides of the formula (4) by the denominator and moving the term $s(z+h)\cdot C(h)$ to the right-hand side, we get the following formula:

$$s(z+h) = A(h)\cdot s(z) + B(h) - s(z+h)\cdot C(h).$$

In particular, if we take three different values $z = z_1$, $z = z_2$, and $z = z_3$, then, for each h, we get the following system of three linear equations for determining the three values $A(h)$, $B(h)$, and $C(h)$:

$$s(z_1+h) = A(h)\cdot s(z) + B(h) - s(z_1+h)\cdot C(h);$$

$$s(z_2+h) = A(h)\cdot s(z) + B(h) - s(z_2+h)\cdot C(h);$$

$$s(z_3+h) = A(h)\cdot s(z) + B(h) - s(z_3+h)\cdot C(h).$$

Due to Cramer's rule, the solution to this system is a ratio of two determinants, i.e., of two polynomials of the coefficients and is, thus, a smooth function of the values $s(z_i+h)$. Since the function $s(z)$ is smooth, we conclude that all three functions $A(h)$, $B(h)$, and $C(h)$ are also smooth. Thus, we can differentiate both sides of the equation (4) by h and get

$$s'(z+h) = \frac{N(h)}{(C(h)\cdot s(z)+1)^2},$$

where

$$N(h) \overset{\text{def}}{=} (A'(h)\cdot s(z) + B'(h))\cdot(C(h)\cdot s(z)+1) -$$

$$(A(h)\cdot s(z) + B(h))\cdot(C'(h)\cdot s(z)).$$

In particular, for $h = 0$, taking into account that $A(h) = 1$ and $B(h) = C(h) = 0$, we conclude that

$$s'(z) = a_0 + a_1\cdot s(z) + a_2\cdot(s(z))^2, \qquad (5)$$

where we denoted $a_0 = B'(0)$, $a_1 = A'(0)$, and $a_2 = -C'(0)$, i.e., $\dfrac{ds}{dz} = a_0 + a_1\cdot s + a_2\cdot s^2$. If we move all the terms related to s to the left-and side and all the terms related to z to the right-hand side, we get the following formula:

$$\frac{ds}{a_0 + a_1\cdot s + a_2\cdot s^2} = dz. \qquad (6)$$

Let us show how we can integrate both sides of this formula and get an explicit expression of $z(s)$, and how based on this expression, we can find the explicit formula for the dependence of s on z.

The generic case is when $a_2 \neq 0$. In this case, we can multiply both sides of the formula (6) by a_2 and get

$$\frac{ds}{\dfrac{a_0}{a_2} + \dfrac{a_1}{a_2}\cdot s + s^2} = a_2\cdot dz. \qquad (7)$$

The quadratic form in the denominator of the left-hand side can be represented as $(s+p)^2 + q$, where $p = \dfrac{a_1}{2a_2}$ and $q = \dfrac{a_0}{a_2} - p^2$. Thus, the formula (7) takes the form

$$\frac{ds}{(s+p)^2 + q} = a_2\cdot dz. \qquad (8)$$

So, for $s_1 = s + p$, we have

$$\frac{ds_1}{s_1^2 + q} = a_2\cdot dz. \qquad (9)$$

Let us now consider all three possible cases: when $q = 0$, when $q > 0$, and when $q < 0$. When $q = 0$, then integrating both sides of (9), we get

$$-\frac{1}{s_1} = a_2\cdot z + c,$$

for some integration constant c, thus $s_1(z) = -\dfrac{1}{a_2\cdot z + c}$ and

$$s(z) = s_1(z) - p = -\frac{1}{a_2\cdot z + c} - p.$$

This function is not everywhere defined – namely, it is not defined for $z = -c/a_2$, thus we will not consider it.

When $q > 0$, then for $s_2 = s_1/\sqrt{q}$, we have $s_1 = s_2 \cdot \sqrt{q}$ thus $ds_1 = \sqrt{q} \cdot ds_2$, $s_1^2 + q = q \cdot s_2^2 + q = q \cdot (s_2^1 + 1)$, and (9) becomes $\frac{1}{\sqrt{q}} \cdot \frac{ds_2}{s^2+1} = a_2 \cdot z$. Integrating leads to

$$\frac{1}{\sqrt{q}} \cdot \arctan(s_2) = a_2 \cdot z + c$$

hence $s_2(z) = \tan(\sqrt{q} \cdot a_2 \cdot z + \sqrt{q} \cdot c)$. This value is not always defined, thus $s_1(z) = \sqrt{q} \cdot s_2(z)$ and $s(z) = s_1(z) - p$ are also not always defined, so we will consider this case either.

For $q < 0$, for $s_2 = s_1/\sqrt{|q|}$, we similarly have $s_1 = s_2 \cdot \sqrt{|q|}$ thus $ds_1 = \sqrt{|q|} \cdot ds_2$, $s_1^2 + q = |q| \cdot s_2^2 + q = |q| \cdot (s_2^2 - 1)$, thus (9) becomes

$$\frac{1}{\sqrt{|q|}} \cdot \frac{ds_2}{s^2-1} = a_2 \cdot z. \qquad (10)$$

One can easily check that

$$\frac{1}{s_2^2 - 1} = \frac{1}{2} \cdot \left(\frac{1}{s_2 - 1} - \frac{1}{s_2 + 1} \right),$$

thus integrating (10), we get

$$\frac{1}{2\sqrt{|q|}} \cdot (\ln(s_2 - 1) - \ln(s_2 + 1)) = a_2 \cdot z + c.$$

Multiplying both sides by $2\sqrt{|q|}$, we conclude that the difference

$$\ln(s_2 - 1) - \ln(s_2 + 1) = \ln \left(\frac{s_2 - 1}{s_2 + 1} \right)$$

is equal to a linear function z_1 of z. Thus, the fractional-linear ratio $\frac{s_2 - 1}{s_2 + 1}$ is equal to $\exp(z_1)$. The inverse to a fractional linear transformation is also fractional linear, so $s_2(z)$ is a fractional linear function of $\exp(z_1)$. The original function $s(z)$ is obtained from $s_2(z)$ by a linear transformation and is, thus, also a fractionally linear expression in terms of $\exp(z_1)$, i.e.,

$$s(z) = \frac{a \cdot \exp(z_1) + b}{c \cdot \exp(z_1) + 1}.$$

For this expression to always be defined, we need $c > 0$; else, if $c < 0$, it is not defined for $z_1 = -\ln(|c|)$. The expression for $s(z)$ can be written as

$$\frac{a}{c} + \text{const} \cdot \frac{1}{c \cdot \exp(z_1) + 1},$$

and for $z_2 = -z_1 + \ln(c)$, as

$$s(z) = \frac{a}{c} + \text{const} \cdot \frac{1}{1 + \exp(-z_2)}.$$

So, each such activation function $s(z)$ can be obtained if we:

- first, apply some linear transformation to z, getting z_2;
- then apply a sigmoid function; and
- finally, we apply a linear transformation to the result.

In the traditional neural network, as we mentioned earlier, we always apply some linear transformation *before* we apply the activation function, and we also apply some linear transformation *after* we apply the activation function. So, from the viewpoint of the above general formula of the traditional neural network, the class of functions which can be represented with K neurons by using the activation function $s(z)$ is exactly the same as the class of functions represented with K neurons by using the sigmoid function. In this sense, *sigmoid is the only shift-invariant activation function* – which explains its efficiency in traditional neural networks.

Limit cases. In the previous subsection, we considered the generic case when $a_2 \neq 0$. To complete our analysis, we need to also consider the remaining case when $a_2 = 0$. This is a limit case of the generic case when $a_2 \to 0$. In this case, the formula (6) takes the following simplified form:

$$\frac{ds}{a_0 + a_1 \cdot s} = dz. \qquad (11)$$

If $a_1 \neq 0$, then for $s_1 = a_0 + a_1 \cdot s$, we have $ds_1 = a_1 \cdot ds$, hence $ds = ds_1/a_1$, and (11) takes the form

$$\frac{1}{a_1} \cdot \frac{ds_1}{s_1} = dz.$$

Integrating, we get $\frac{1}{a_1} \cdot \ln(s_1) = z + c$, hence $\ln(s_1) = z_1 \overset{\text{def}}{=} a_1 \cdot z + a_1 \cdot c$, so $s_1(z) = \exp(z_1)$, and $s(z) = \frac{s_1(z) - a_0}{a_1} = \frac{1}{a_1} \cdot (\exp(z_1) - a_0)$. The resulting activation functions $s(z)$ can be obtained if we:

- first, apply some linear transformation to z, getting z_2;
- then apply an exponential function; and
- finally, we apply a linear transformation to the result.

Similarly to the generic case, we can thus conclude that the class of functions which can be represented with K neurons by the using the activation function $s(z)$ is exactly the same as the class of functions represented with K neurons by using the exponential function.

The only remaining case if $a_1 = 0$. In this case, (11) easily integrates into $\frac{s}{a_0} = z + c$, i.e, to a linear activation function $s(z) = a_0 \cdot z + a_0 \cdot c$. Mathematically, it is a legitimate case, but, of course, from the viewpoint of neural networks it makes no sense, since, as we have mentioned earlier, the whole point of activation functions is to cover *non*-linear functions.

IV. Why Squashing Activation Functions

We need multi-layer neural networks. The problem with traditional neural networks, as we mention in [**?**], [1], [7], is that they waste a lot of bits: for K neurons, any of $K!$ permutations results in exactly the same function. To decrease this duplication, we need to decrease the number of neurons K in each layer. So, instead of placing all nonlinear neurons in one layer, we place them in several consecutive layers. This is one of the main idea behind deep learning.

Which activation function should we use: analysis of the problem. In the first nonlinear layer, we make sure that a shift in the input – corresponding to a different estimate of the background noise – does not change the processing formula, i.e., that results $s(z+c)$ and $s(z)$ can be obtained from each other by applying an appropriate transformation – in this case, a fractional-linear transformation. We already know that this idea leads either to the sigmoid function (or to its limit case – exponential function).

So far, so good, but this logic does not work if we try to find out what activation function we should use in the *next* nonlinear layers. Indeed, the output of the first layer – which is the input to the second nonlinear layer – is *no longer shift*-invariant, it is invariant with respect to some more *complex* (fractional linear) transformations. We know what to do when the input is shift-invariant, so a natural idea is to perform some *additional* transformation that will make the results shift-invariant. If we do that, then we will again be able to apply the sigmoid activation function, then again the additional transformation, etc.

These additional transformations should transform generic fractional-linear operations into shift. This means that the inverse of such a transformation should transform shifts into some fractional-linear operations. But this is exactly what we analyzed in the previous section – transformations that transform shifts into fractional-linear operations. We already know the formulas $s(z)$ for these transformations. In general, they are formed as follows:

- first, we apply some linear transformation to the input z, resulting in a linear combination $Z = p\cdot z + q$;
- then, we compute $Y = \exp(Z)$; and
- finally, we apply some fractional-linear transformation to the resulting value Y, getting y.

So, to get the inverse transformation, we need to reverse all three steps, starting with the last one:

- first, we apply a fractional-linear transformation to y, getting Y;
- then, we compute $Z = \ln(Y)$; and
- finally, we apply a linear transformation to Z, resulting in z.

This leads exactly to squashing functions. What happens if we first apply some sigmoid-type transformation moving us from shifts to tractional-linear operations and then an inverse-type transformation? The last step of the sigmoid-type transformation and the first step of the inverse-type transformation both apply fractional-linear transformations. Since the composition of fractional-linear transformations is fractional-linear, we can combine them into a single step. Thus, the resulting combined activation function can thus be described as follows:

- first, we apply some linear transformation L_1 to the input z, resulting in a linear combination $Z = L_1(z) = p\cdot z+q$;
- then, we compute $E = \exp(Z) = \exp(L_1(z))$;
- then, we apply some fractional-linear transformation F to $E = \exp(Z)$, getting $T = F(E) = F(\exp(L_1(z))$;
- then, we compute $Y = \ln(T) = \ln(F(\exp(L_1(z)))$; and
- finally, we apply a linear transformation L_2 to Y, resulting in the final value $y = s(z) = L_2(Y) = L_2(\ln(F(\exp(L_1(z))))$.

One can check that these are exactly squashing function! Thus, squashing functions can indeed be naturally explained by the invariance requirements.

Example. As an example of the above description, let us provide a family of squashing functions that tend to the rectified linear activation function $\max(z, 0)$. For this purpose, let us take:

- $L_1(z) = k\cdot z$, with $k > 0$, so that $E = \exp(L_1(z)) = \exp(k\cdot z)$;
- $F(E) = 1 + E$, so that $T = F(E) = \exp(k\cdot z) + 1$ and $Y = \ln(T) = \ln(\exp(k\cdot z) + 1)$; and
- $L_2(Y) = \dfrac{1}{k}\cdot Y$, so that the resulting activation function takes the form $s(z) = \dfrac{1}{k}\cdot\ln(\exp(k\cdot z)+1)$.

Let us show that this expression tends to the rectified linear activation function when $k\to\infty$.

When $z < 0$, then $\exp(k\cdot z)\to 0$, so $\exp(k\cdot z)+1\to 1$, $\ln(\exp(k\cdot z)+1)\to 0$ and so $s(z)\to 0$.

On the other hand, when $z > 0$, then

$$\exp(k\cdot z)+1 = \exp(k\cdot z)\cdot(1+\exp(-k\cdot z)),$$

thus $\ln(\exp(k\cdot z)+1) = k\cdot z + \ln(1+\exp(-k\cdot z))$ and

$$s(z) = \frac{1}{k}\cdot\ln(\exp(k\cdot z)+1) = z + \frac{1}{k}\cdot\ln(1+\exp(-k\cdot z)).$$

When $k\to\infty$, we have $\exp(-k\cdot z)\to 0$, hence

$$1+\exp(-k\cdot z)\to 1,$$

$\ln(1+\exp(-k\cdot z))\to 0$, so $\dfrac{1}{k}\cdot\ln(1+\exp(-k\cdot z))\to 0$ and indeed $s(z)\to z$.

References

[1] 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, Berlin, Heidelberg: Springer Verlag, 2018, pp. 1–6.

[2] O. Csiszár, G. Csiszár, and J. Dombi, Interpretable Neural Networks Based on Continuous-Valued Logic and Multicriterion Decision Operators, arXiv:1910.02486v2, posted on February 7, 2020.

[3] J. Dombi and O. Csiszár, "Operator-dependent modifiers in nilpotent logical systems", Proceedings of the 10th International Joint Conference on Computational Intelligence IJCCI'2018, Seville, Spain, September 18–20, 2018, pp. 126–134.

[4] J. Dombi and Zs. Gera, "The approximation of piecewise linear membership functions and Lukasiewicz operators", Fuzzy Sets and Systems, vol. 154, pp. 275–286, 2005.

[5] O. Fuentes, J. Parra, E. Anthony, and V. Kreinovich, "Why rectified linear neurons are efficient: a possible theoretical explanations", In: O. Kosheleva et al. (eds.), Beyond Traditional Probabilistic Data Processing Techniques, Cham, Switzerland: Springer, 2020, to appear.

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

[7] V. Kreinovich and O. Kosheleva, "Deep learning (partly) demystified", Proceedings of the 4th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence ISMSI'2020, Thimpu, Bhutan, March 21–22, 2020, to appear.

[8] V. Kreinovich and C. Quintana. "Neural networks: what non-linearity to choose?," Proc. 4th Univ. of New Brunswick AI Workshop, Fredericton, New Brunswick, Canada, 1991, pp. 627–637.

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

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