Why Squashing Functions in Multi-Layer Neural Networks

Julio C. Urenda[1], Orsolya Csiszár[2,3], Gábor Csiszár[4], József Dombi[5], Olga Kosheleva[1], Vladik Kreinovich[1], György Eigner[3]

[1]University of Texas at El Paso, USA
[2]University of Applied Sciences Esslingen, Germany
[3]Óbuda University, Budapest, Hungary
[4]University of Stuttgart, Germany
[5]University of Szeged, Hungary

E-mails: vladik@utep.edu, orsolya.csiszar@nik.uni-obuda.hu, gabor.csiszar@mp.imw.uni-stuttgart.de, dombi@inf.u-szeged.hu, olgak@utep.edu, vladik@utep.edu, eigner.gyorgy@nik.uni-obuda.hu

1. A Short Introduction

- In their successful applications, deep neural networks use a non-linear transformation $s(z) = \max(0, z)$.
- It is called a *rectified linear* activation function.
- Sometimes, more general transformations – called *squashing functions* – lead to even better results.
- In this talk, we provide a theoretical explanation for this empirical fact.
- To provide this explanation, let us first briefly recall:
  - why we need machine learning in the first place,
  - what are deep neural networks, and
  - what activation functions these neural networks use.

2. Machine Learning Is Needed

- 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.
- So, the resulting model of the whole system is too inaccurate to be useful.
- We also need to use the records of the actual system's behavior when making predictions.
- Using the previous behavior to predict the future is called *machine learning*.

3. Deep Learning

- The most efficient machine learning technique is *deep learning*: the use of multi-layer neural networks.
- In general, on a layer of a neural network, we transform signals $x_1, \ldots, x_n$ into a new signal $y = s\left(\sum_{i=1}^{n} w_i \cdot x_i + w_0\right)$.
- The coefficient w_i (called *weights*) are to be determined during training.
- $s(z)$ is a non-linear function called *activation function*.
- Most multi-layer neural networks use $s(z) = \max(z, 0)$ known as *rectified linear* function.

4. Shall We Go Beyond Rectified Linear?

- Preliminary analysis shows that for some applications:
  - it is more advantageous to use different activation functions for different neurons;
  - specifically, this was shown for a special family of *squashing* activation functions

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

  - this family contains rectified linear neurons as a particular case.
- We explain their empirical success of squashing functions by showing that:
  - their formulas
  - follow from reasonably natural symmetries.

5. How This Talk Is Structured

- First, we recall the main ideas of symmetries and invariance.
- Then, we recall how these ideas can be used to explain the efficiency of the sigmoid activation function

$$s_0(z) = \frac{1}{1 + \exp(-z)}.$$

- This function is used in the traditional 3-layer neural networks.
- Finally, we use this information to explain the efficiency of squashing activation functions.

6. Which Transformations Are Natural?

- From the mathematical viewpoint, we can apply any non-linear transformation.
- However, some of these transformations are purely mathematical, with no clear physical interpretation.
- Other transformation are *natural* in the sense that they have physical meaning.
- What are natural transformations?

7. 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.
- However, from the physical viewpoint, the numerical values are not absolute; they change:
  - if we change the measuring unit and/or
  - the starting point for measuring the corresponding quantity.
- The corresponding changes in numerical values are clearly physically meaningful, i.e., natural.
- For example, we can measure a person's height in meters or in centimeters.

8. Numerical Values Change (cont-d)

- 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:
  - instead of the original numerical value x,
  - we get a new numerical value $\lambda \cdot x$ – while the actual quantity remains the same.
- Such a transformation $x \to \lambda \cdot x$ is known as *scaling*.
- For some quantities, e.g., for time or temperature, the numerical value also depends on the starting point.
- For example, we can measure the time from the moment when the talk started.
- Alternatively, we can use the usual calendar time, in which Year 0 is the starting point.

9. Numerical Values Change (cont-d)

- 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 $x + x_0$.
- Such a transformation $x \to x + x_0$ is known as *shift*.
- In general, if we change both the measuring unit and the starting point, we get a linear transformation:

$$x \to \lambda \cdot x + x_0.$$

- A usual example of such a transformation is a transition from Celsius to Fahrenheit temperature scales:

$$t_F = 1.8 \cdot t_C + 32.$$

10. 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.

11. Invariance (cont-d)

- 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.

12. Natural Transformations Beyond Linear Ones

- In some cases, the relation between different scales is non-linear.
- For example, we can measure the earthquake energy:
  - in Joules (i.e., in the usual scale) or
  - in a logarithmic (Richter) scale.
- Which nonlinear transformation are natural?
- First, as we have argued, 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.

13. Natural Transformations (cont-d)

- 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.
- The class of transformations that satisfies the 2nd and 3rd properties is called a *transformation group*.
- 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 should be determined by a finite number of parameters.

14. Natural Transformations (cont-d)

- The smallest number of parameters needed to describe a family is known as the *dimension* of this family.
- E.g., 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.

15. 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*, 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.
- This phase corresponds to the group of all possible continuous transformations.

16. All Natural Transformations (cont-d)

- 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.
- These are linear (= affine) transformations.
- When we get even closer, we become able to detect the shapes, sizes, etc.

17. All Natural Transformations (cont-d)

- Wiener argued that there are no other transformation groups – since:
  - if there were other transformation groups,
  - after billions years of evolution, we would use them.
- In precise terms, he conjectured that:
  - the only finite-dimensional transformation group that contain all linear transformations
  - is the groups of all projective transformations.
- This conjecture was later proven.
- For transformations of the real line, projective transformations are simply fractional-linear transformations

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

- So, natural transformations are fractional-linear ones.

18. Traditional Neural Networks (NN)

- Let us recall why traditional neural networks appeared in the first place.
- 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.
- Then, 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.

19. Traditional Neural Networks (cont-d)

- 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.

20. Traditional Neural Networks (cont-d)

- 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; we will call such a layer a *d-layer*; d for *d*evice.
- Then, the results of this d-layer go to another d-layer, etc.
- The fewer d-layers we have, the faster the computations.

21. How Many d-Layers Do We Need?

- It can be proven that:
  - 1-d-layer schemes (L or NL) are not sufficient to approximate any possible dependence, and
  - 2-d-layer schemes (L-NL, linear layer followed by non-linear layer, or NL-L) are also not enough.
- Thus, we need at least 3-d-layer networks – and 3-d-layer networks can be proven to be sufficient.
- In a 3-d-layer network:
  - we cannot have two linear layers or two nonlinear d-layers following each other,
  - that would be equivalent to having one d-layer since, e.g., a composition of two L functions is also L.
- So, our only options are L-NL-L and NL-L-NL.

22. How Many d-Layers Do We Need (cont-d)

- Since linear transformations are faster to compute, the fastest scheme is L-NL-L.
- In this scheme:
  - first, each neuron k in the L d-layer combines the inputs into a linear combination

$$z_k = \sum_{i=1}^{n} w_{ki} \cdot x_i + w_{k0};$$

  - then, in the next d-layer, each such signal is transformed into $y_k = s_k(z_k)$ for some non-linear f-n;
  - finally, in the last linear d-layer, we form a linear combination of the values y_k: $y = \sum\limits_{k=1}^{K} W_k \cdot y_k + W_0$.

23. How Many d-Layers Do We Need (cont-d)

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

24. Traditional NN Mostly Used Sigmoid

- Originally, the sigmoid function was selected because:
  - it provides a reasonable approximation to
  - how biological neurons process their inputs.
- Several other nonlinear activation functions have been tried.
- However, in most cases, the sigmoid $s_0(z)$ leads to the best approximation results.
- A partial explanation for this empirical success is that:
  - neural networks using sigmoid activation function $s_0(z)$ have proven to be universal approximators;
  - i.e., the corresponding neural networks can approximate any continuous function.
- However, many other non-linear activation functions have the same universal approximation property.

25. So, Why Sigmoid?

- We have mentioned that the values of physical quantities change when we:
  - change the starting point,
  - i.e., shift all the data points by the same constant x_0.
- At first glance, it may seem that this does not apply to neural data processing, since usually:
  - before we apply a neural network,
  - we *normalize* the data, i.e., transform all the input values into the some fixed interval (e.g., $[0, 1]$).
- This normalization is based on all the values of the corresponding quantity that have been observed so far.
- The smallest of these values corresponds to 0 and the largest to 1.

26. Why Sigmoid (cont-d)

- However, as we will show, shift still makes sense even for the normalized data.
- Indeed, in real life, signals come with noise, in particular, with background noise.
- Often, a significant part of this noise is a constant which is added to all the measured signals.
- This constant noise component is, in general, different for different situations.
- We can try to get rid of this constant noise component by subtracting the corresponding constant.
- So, we replace:
  - each original numerical value x_i
  - with a corrected value $x_i - n_i$.

27. Why Sigmoid (cont-d)

- After this correction, instead of the original value z_k, we get a corrected value

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

- Here, we denoted $h'_k \stackrel{\text{def}}{=} \sum\limits_{i=1}^{n} w_{ki} \cdot n_i$.
- The trouble is that we do not know the exact values of these constants n_i.
- So, depending on our estimates, 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 $h_k \stackrel{\text{def}}{=} h'_k - h''_k$: $z''_k = z'_k + h_k$.

28. Why Sigmoid (cont-d)

- This is 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,
  - i.e., this formula should be, in this sense, *shift-invariant.*
- In other words:
  - if we start with the formula $y = s(z)$ and we shift from z to $z' = z + h$,
  - then we should have the same relation $y' = s(z')$ for an appropriately transformed $y' = f(y)$.

29. Why Sigmoid (cont-d)

- For different shifts h, we will have, in general, different natural transformations $f(y)$.
- We have mentioned that all natural transformations $f(y)$ are fractionally linear.
- Thus, for each h, $y' = s(z+h)$ should be fractional-linear in $y = s(z)$:
$$s(z+h) = \frac{a(h) \cdot s(z) + b(h)}{c(h) \cdot s(z) + d(h)}.$$
- It turns out that this implies the sigmoid $s_0(z)$.

30. Why Sigmoid: Derivation

- 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 above formula by $d(h)$ and get a simpler formula:

$$s(z+h) = \frac{A(h) \cdot s(z) + B(h)}{C(h) \cdot s(z) + 1}, \text{ where } A(h) = a(h)/d(h), \ldots$$

- 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 defined and smooth for all z.

31. Why Sigmoid: Derivation (cont-d)

- Indeed, on each interval, every continuous function:
  - can be approximated, with any desired accuracy,
  - by a smooth one – even by a polynomial.
- So, from the practical viewpoint, it is sufficient to only consider smooth activation functions.
- Multiplying both sides of the above formula by the denominator, we get:

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

- Let us take three different values z_i.
- Then, for each h, we get 3 linear equations for three unknown $A(h)$, $B(h)$, and $C(h)$:

$$s(z_i+h) = A(h)\cdot s(z_i)+B(h)-C(h)\cdot s(z_i+h)\cdot s(z_i),\ i=1,2,3.$$

32. Why Sigmoid: Derivation (cont-d)

- Due to Cramer's rule, the solution to this system is:
  - a ratio of two determinants,
  - i.e., a ration of two polynomials of the coefficients.
- Thus, $A(h)$, $B(h)$, and $C(h)$ are smooth functions 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 above equation by h and get

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

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

33. Why Sigmoid: Derivation (cont-d)

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

  $$s'(z) = a_0 + a_1 \cdot s(z) + a_2 \cdot (s(z))^2, \text{ where } a_0 = B'(0), \ldots$$

- So, $\dfrac{ds}{dz} = a_0 + a_1 \cdot s + a_2 \cdot s^2$ and

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

- We can now integrate both sides of this formula and get an explicit expression of $z(s)$.
- Based on this expression, we can find the explicit formula for the dependence of s on z.

34. Why Sigmoid: Derivation (cont-d)

- The only non-linear dependencies $s(z)$ are:
  - the sigmoid (plus some linear transformations before and after) and
  - the sigmoid's limit case $\exp(z)$.
- So, the sigmoid $s_0(z)$ is the only shift-invariant activation function.
- This explains its efficiency in traditional neural networks.

35. We Need Multi-Layer Neural Networks

- The problem with traditional neural networks 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.

36. Which Activation Function Should We Use

- In the first nonlinear d-layer, we make sure that:
  - a shift in the input – corresponding to a different estimate of the constant noise component,
  - does not change the processing formula,
  - i.e., that results $s(z+c)$ and $s(z)$ can be obtained from each other by an appropriate transformation.
- We already know that this idea leads to the sigmoid function $s_0(z)$.
- This logic doesn't work if we try to find out what activation function we should use in the *next* NL d-layer.
- Indeed, the input to the 2nd NL d-layer is the output of the 1st NL d-layer.
- This input is *no longer* shift-invariant.

37. Which Activation Function (cont-d)

- This input 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 $s_0(z)$,
  - then again the additional transformation, etc.
- These additional transformations should transform generic fractional-linear operations into shift.

38. Which Activation Function (cont-d)

- Thus, the inverse of such a transformation should transform shifts into fractional-linear operations.
- But this is exactly what we analyzed earlier – transformations that transform shifts into fractional-linear.
- 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.

39. Which Activation Function (cont-d)

- 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.

40. This Leads Exactly to Squashing Functions

- What happens if we:
  - first apply a sigmoid-type transformation moving us from shifts to tractional-linear operations,
  - and then an inverse-type transformation?
- The last step of the sigmoid transformation and the first step of the inverse are fractional-linear.
- The composition of fractional-linear transformations is fractional-linear.
- So, we can combine these 2 steps into a single step.

41. This Leads to Squashing Functions (cont-d)

- 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 a 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

$$y = s(z) = L_2(Y) = L_2(\ln(F(\exp(L_1(z))))).$$

42. This Leads to Squashing Functions (cont-d)

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

43. Example

- 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 $s(z) \to 0$.

44. Example (cont-d)

- 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

$$1 + \exp(-k \cdot z) \to 1, \quad \ln(1 + \exp(-k \cdot z)) \to 0.$$

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

45. Acknowledgments

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 (Preparing a New Generation for Professional Practice in Computer Science),
  - HRD-1242122 (Cyber-ShARE Center of Excellence);
- by the European Research Council (ERC):
  - under the European Union's Horizon 2020 Research and Innovation Programme,
  - grant agreement No. 679681.