How to Make AI More Reliable

Olga Kosheleva[1] and Vladik Kreinovich[2]

Departments of [1]Teacher Education and [2]Computer Science
University of Texas at El Paso, 500 W. University
El Paso, Texas 79968, USA
olgak@utep.edu, vladik@utep.edu

1. Formulation of the problem

- Modern machine learning techniques such as deep learning have been spectacularly successful.
- They enabled computer systems to beat the world champion in Go.
- They enable computer systems line ChatGPT to have meaningful conversations and to answer complex questions.
- However, these techniques are not absolutely reliable: as most other techniques, they sometimes lead to wrong results.
- This is especially important in engineering applications such as self-driving cars:
  - in spite of great promises, they turned to be even less reliable than human drivers,
  - often making simple mistakes that even a mildly experienced human driver would never make.

2. Formulation of the problem (cont-d)

- At first glance, this may not sound as a serious problem: no methods are perfect.
- But from the viewpoint of reliability, there is an important difference between traditional methods and deep learning.
- Traditional methods are usually based on well-developed well-analyzed techniques such as probability and statistics.
- The known mathematical foundations for the traditional techniques make them more reliable.
- We can gauge the corresponding uncertainty.
- We can analyze how the uncertainty in input data affects the uncertainty of the data processing results, etc.

3. Formulation of the problem (cont-d)

- In contrast, many AI techniques – such as deep learning techniques – are heuristic.
- They lack theoretical foundations.
- They work on several examples.
- However, because of the lack of foundations, there is no guarantee that they will work in other cases as well.

4. What we do in this talk

- We analyze heuristic, mostly empirically selected features of modern AI techniques, such as:
  - deep learning, and
  - fuzzy techniques – translating natural-language expert knowledge into computer-understandable mathematical terms.
- We show that these techniques can be mathematically justified – which makes them more reliable.
- For example, this justification explains the choice of ReLU activation function, etc.
- Moreover, we show that natural ideas lead only to these two classes of techniques: fuzzy and deep learning.
- Our explanation is based on the following two main ideas:

5. What we do in this talk (cont-d)

- The first idea is related to computational complexity.
- Namely, it is related to the fact that it is usually possible:
  - to improve the quality of a computation result
  - by performing additional computations.
- Our computational resources are limited.
- So, we need to select computational schemes in which each elementary computation requires the smallest possible number of resources.
- The second idea is the idea of natural symmetries, an idea actively used in physics.

6. Computational complexity: main idea

- Often – e.g., for an iterative algorithm – the more computations we perform, the more accurate and reliable is the result.
- Thus, to make data processing results more accurate and reliable, it is desirable to perform as many computational steps as possible.
- Since the overall number of computational steps is usually limited, this means making elementary computational steps as fast as possible.

7. Which computational steps are the fastest?

- All computations on a computer consists of a sequence of hardware supported operations:
  - computing minimum and maximum,
  - addition and subtraction, and
  - computing multiplication.
- Division is usually not directly hardware supported, it is performed as a sequence of several multiplications and additions.
- Which of these operations are faster and which are slower?

8. How fast is computing min and max?

- et us start with computing min and max, i.e., in effect, with finding out which of the two numbers is larger.
- If we have two binary numbers, then in half of the cases their first bits are different.
- In this case we know which of the two numbers is larger: the number that starts with 1 is larger than the number that starts with 0.
- So, with probability 1/2, we only need 1 bit operation.
- In the remaining half of the cases, the first bits are equal.
- Then, if the second bits are different:
  - we also know which number is larger:
  - the number for which the second bit is 1 is larger than the number whose second bit is 0.

9. How fast is computing min and max (cont-d)

- One can check that the probability of this is 1/4:
  - with probability 1/2, the first bit of the second number is the same as the first bit of the first number, and
  - with probability 1/2, the second bits are different.
- So, in this case, with probability 1/4, we need 2 bit operations.
- Similarly, with probability 1/8, we encounter a situation in which the first two bits are the same while the third bits differ.
- In this case, we can also determine which number is larger.
- So, in this case, we need 3 bit operation.
- In general, with probability $1/2^k$, we encounter a situation in which the first $k-1$ bits are the same while the k-th bits differ.
- Based on these bits, we can also determine which number is larger.
- So, in this case, we need k bit operation.

10. How fast is computing min and max (cont-d)

- Thus, the expected number of bit operation to compute min and max is equal to

$$b_m = \frac{1}{2} \cdot 1 + \frac{1}{2^2} \cdot 2 + \frac{1}{2^3} \cdot 3 + \ldots + \frac{1}{2^k} \cdot k + \ldots$$

- To compute this sum, let us multiply all the terms in the formula by 2. Then, we get:

$$2b_m = 1 + \frac{1}{2} \cdot 2 + \frac{1}{2^2} \cdot 3 + \ldots + \frac{1}{2^{k-1}} \cdot k + \frac{1}{2^k} \cdot (k+1) + \ldots$$

- If we subtract these formulas, then in this difference, for each k, the term proportional to $1/2^k$ is equal to

$$\frac{1}{2^k} \cdot (k+1) - \frac{1}{2^k} \cdot k = \frac{1}{2^k}.$$

- Thus, the difference is equal to:

$$2b_m - b_m = b_m = 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots + \frac{1}{2^k} + \ldots$$

11. How fast is computing min and max (cont-d)

- This is a particular case $q = 1/2$ of the geometric progression $1 + q + q^2 + \ldots$ whose sum is well known: it is $1/(1-q)$.
- For $q = 1/2$, we get

$$b_m = \frac{1}{1 - 1/2} = \frac{1}{1/2} = 2.$$

- Thus, computing min and max require, on average, 2 bit operations.

12. How fast are addition and multiplication?

- Adding two n-bit numbers requires:
  - in the best case, n bit operations, and
  - in the worst case, $2n$ bit operations – since we may also need to add the carry bits.
- Thus, a reasonable estimate for the average number of bit operations is
$$\frac{n + 2n}{2} = 1.5 \cdot n.$$
- Binary multiplication consists of adding shifted first number for all the cases when the second number has bit 1.
- On average, half of the bits are 0s and half are 1s.
- So, for multiplying two n-bit numbers, on average, we need $n/2$ additions.
- Since each addition requires, on average, $1.5 \cdot n$ operations, overall we need $\frac{n}{2} \cdot 1.5 \cdot n = 0.75 \cdot n^2$ bit operations.

13. Conclusion: which operations are faster

- Based on the above analysis, we conclude that:
  - the fastest operations are min and max; they require, on average, 2 bit operations;
  - the next fastest is addition; it requires $c \cdot n$ bit operations for n-but numbers; and
  - the slowest is multiplication; it requires $c \cdot n^2$ bit operations.

14. So which operations should we use?

- It would be nice to use only the fastest operations – min and max.
- However, the problem is that these operations always return one of the inputs,
- So, if we start with N data points, all we get is one of these points back.
- Thus, we cannot do any serious data processing.
- So, to be able to perform more general algorithms, we need to also use some other operations.
- The fastest of these other operations is addition.
- We can add the same number several times, so, in effect, we can compute a function $x \mapsto c \cdot x$ for some constant c.

15. So which operations should we use (cont-d)

- Thus, in general, using addition means, in effect, that we can compute:
  - an arbitrary linear combination of inputs $x_1, \ldots, x_m$,
  - i.e., any expression of the type $c_0 + c_1 \cdot x_1 + \ldots + c_m \cdot x_m$.
- If we can compute min, max, and linear combination, then we can compute any function; indeed:
  - this way, we can compute any piece-wise linear function consisting of finitely many linear pieces, and
  - piece-wise functions consisting of finitely many linear pieces are universal approximators.

16. So which operations should we use (cont-d)

- The proof of this universal approximation property is straightforward:
  - We select a dense grid.
  - On each vicinity of a grid, we approximate the function by a constant.
  - Then, we add almost-vertical connections between these constants.
- So, *to perform data processing as fast as possible, we need to limit ourselves to operations* min, max, *and linear combinations.*

17. Additional restrictions

- We can have a linear combination of several inputs. Similarly, we can have min and max of several inputs.
- The more inputs we have, the more time the corresponding operation takes.
- So, to make computations faster, it make sense to restrict the number of inputs to which each of these operations can be applied.
- In principle, we have three options.
- We can:
  - restrict the number of inputs to a linear combination to the smallest possible number – 1 input, and
  - allow min and max of many inputs.

18. Additional restrictions (cont-d)

- We can:
  - restrict the number of inputs in min and max operation to the smallest possible number – 2 inputs, and
  - allow linear combination of many inputs.
- We can also limit both the number of inputs both for linear combination and for min and max to some reasonable numbers.
- Let us analyze what we get in these three cases.

19. What if we restrict the number of inputs to linear combinations: this naturally leads to fuzzy techniques

- In this case, on each computational step:
  - we either compute a linear function of one of the inputs,
  - or we compute min and max of several inputs.
- In particular, if we apply min and max to different linear functions of the same variable, we get general piece-wise linear functions.
- From the mathematical viewpoint, this corresponds to *fuzzy techniques*:
  - where we use piece-wise linear functions (known as *membership function*) whose values are interpreted as degrees of confidence,
  - and where min and max are interpreted as, correspondingly, "and" and "or".
- Thus, fuzzy techniques indeed naturally appear.

20. Comments

- A minor issue is that in the traditional fuzzy logic, values are limited to the interval $[0, 1]$.
- However, this is not a major issue.
- We could as well:
  - take values from any interval, such as $[-1, 1]$ or $[0, 10]$,
  - when we base fuzzy degree of the result of estimation of a 0-to-10 scale.
- Any two intervals can be easily transformed into each other by a linear transformation.
- It should be mentioned that in the past:
  - more general membership functions have been proposed,
  - as well as "and"- and "or"-operations which are more complex than min and max.

21. Comments (cont-d)

- However, in most practical applications, piece-wise membership functions and min and max operations work the best.
- Our analysis explains why.
- It should also be mentioned that often, fuzzy techniques are used to process expert data, this is what they were invented for.
- However, somewhat mysteriously:
  - these techniques have also been successfully used to process data
  - in situations when we do not have any expert knowledge.
- Our analysis explains this mystery: fuzzy techniques naturally appear if we want to make our computations as fast as possible.

22. Comments (cont-d)

- Our analysis also explains:
  - which piece-wise membership functions should be most effective:
  - the ones that require the smallest number of min and max operations.
- And indeed, the most effective are triangular and trapezoidal membership functions.
- They require, correspondingly:
  - one or two min operations between linear functions
  - plus one operation $\max(0, z)$ to make sure that the degree of confidence does not go below 0.
- From this viewpoint, going from triangular to linear functions does not decrease the number of needed min operations.
- Indeed, we will then need to use $\min(1, z)$ to make sure that the degree of confidence does not go above 1.

23. What if we restrict the number of inputs to min and max

- In this case, in one computational step, we compute either a linear combination or min (max).
- Of course, many real-life dependencies are non-linear, so we need a sequence of such operations.
- Let us see what will be computed as a result of two consequent operations.
- If a linear combination is followed by a linear combination, the result is also a linear combination.
- So it can be computed in a single operation.
- Thus, the sequence of two operations only makes sense if a linear combination is followed by min or max.
- We restricted ourselves to only min and max of two inputs.

24. What if we restrict the number of inputs to min and max (cont-d)

- So, as a result of two operations, we can compute the expressions of the type

$$\max(c_0 + c_1 \cdot x_1 + \ldots + c_m \cdot x_m, c'_0 + c'_1 \cdot x_1 + \ldots + c'_m \cdot x_m).$$

- We can also get similar operations with min instead of max
- Which number is larger does not change if we add the same amount to both values.
- Thus, in general, $\max(a + c, b + c) = c + \max(a, b)$.
- In particular, we can take $a = 0$ and

$$c = c_0 + c_1 \cdot x_1 + \ldots + c_m \cdot x_m, \text{ and } b = (c'_0 - c_0) + (c'_1 - c_1) \cdot x_1 + \ldots + (c'_m - c_m) \cdot x_m.$$

- Then we can represent the above expression in the following equivalent form:

$$c_0 + c_1 \cdot x_1 + \ldots + c_m \cdot x_m + \max(0, (c'_0 - c_0) + (c'_1 - c_1) \cdot x_1 + \ldots + (c'_m - c_m) \cdot x_m).$$

25. What if we restrict the number of inputs to min and max (cont-d)

- This is, in effect, exactly what is computed by a single neuron with so-called ReLU activation function $s(z) = \max(0, z)$.
- In general, the output signal of a neuron has the form

$$y = s(c_0 + c_1 \cdot x_1 + \ldots + c_m \cdot x_m).$$

- Also, we can always add linear combinations.
- Vice versa, any result of a neuron with ReLU activation function has the above form.
- Let us consider a similar expression with min:

$$\min(c_0 + c_1 \cdot x_1 + \ldots + c_m \cdot x_m, c'_0 + c'_1 \cdot x_1 + \ldots + c'_m \cdot x_m).$$

- We can use a similar fact that $\min(a, b) = a - \max(0, b - a)$ and thus, get the equivalent expression

$$c_0 + c_1 \cdot x_1 + \ldots + c_m \cdot x_m - \max(0, (c'_0 - c_0) + (c'_1 - c_1) \cdot x_1 + \ldots + (c'_m - c_m) \cdot x_m).$$

26. What if we restrict the number of inputs to min and max (cont-d)

- Thus, we also get the result of a ReLU-based neuron.
- So, *this restriction naturally leads to deep learning with ReLU neurons.*
- Similarly to the fuzzy case:
  - in principle, we can consider – and have considered – different activation functions,
  - but ReLU seem to be the most efficient.
- Our analysis explains why.

27. What if we restrict the number of inputs for both operations: this naturally leads to convolutional neural networks

- In this case, each computational step includes either linear combination of a few inputs or min or max of a few inputs.
- This arrangement corresponds to Convolutional Neural Networks, where on each computation layer, we:
  - either compute convolution – i.e., for each result, a linear combination of a few previous results,
  - or max-pooling, where we compute the max (or min) of several previous results.

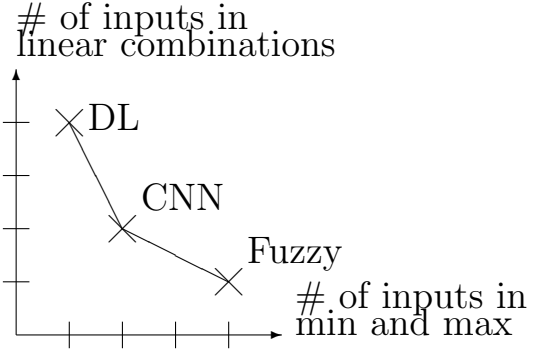
28. Summarizing

- A requirement to have computational steps as fast as possible leads to 3 computational schemes: fuzzy, deep neural networks (DNN), and convolutional neural networks (CNN).
- This result explains why these three computational schemes have indeed been very successful.
- Specifics of how each of these schemes appear can be summarized by the following table:

Number of inputs to min and max	Number of inputs to linear combinations	Type of computation system
unlimited	minimal	fuzzy techniques
restricted	restricted	convolutional neural networks
minimal	unlimited	deep learning with ReLU activation function

29. Summarizing (cont-d)

This can also be described graphically:



30. Which of these computation schemes is faster?

- As we have mentioned, computing min and max is much faster than computing linear combinations.
- Thus, the more restrictions we place on linear combinations, the faster the computations.
- From this viewpoint:
  - fuzzy techniques are the fastest,
  - convolutional neural networks are medium fast, and
  - deep learning techniques are the slowest.
- This conclusion perfectly fits with observations.

31. Which of these computation schemes is faster (cont-d)

- Computation speed is still one of the main obstacles to more effective computations.
- So, it is not surprising that:
  - historically, fuzzy techniques appeared first,
  - convolutional schemes – e.g., for image processing – also appeared some time ago, while
  - deep learning techniques appeared much later.

32. Which of these computation schemes leads to the most accurate results?

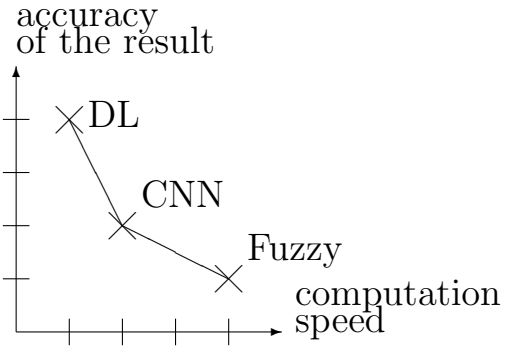
- As we have mentioned earlier, min and max does not perform any real data processing, they just select one of the given inputs.
- To perform real data processing, to go beyond the input values, we thus need to use linear combinations.
- So, the fewer restrictions we put on linear combinations:
  - the more data processing we can perform,
  - the more accurate results we will attain.
- From this viewpoint:
  - fuzzy techniques are the least accurate,
  - convolutional neural networks are medium accurate, and
  - deep learning techniques are the most accurate.
- This conclusion also perfectly fits with observations.

33. Summarizing

Type of computation system	Computation speed	Accuracy of the result
fuzzy techniques	fastest	least accurate
convolutional neural networks	medium fast	medium accuracy
deep learning with ReLU activation function	slowest	most accurate

34. Summarizing (cont-d)

These comparison results can also be described graphically:



35. Auxiliary issue: which "and"- and "or"-operations beyond min and max should we choose?

- As we have mentioned, the main purpose of the expert-based applications of fuzzy logic is to translate:
  - expert rules formulated by using imprecise ("fuzzy") words from natural language
  - into precise control strategy.
- Some expert rules have simple conditions.
- For example, an expert may say: "If you are driving a little bit above speed limit, brake a little bit".
- To describe such rules, a natural idea is:
  - to elicit, from the expert,
  - his/her degree of confidence – from the interval $[0, 1]$ – to which, e.g., 5 mph above speed limit is "a little bit".
- This may be somewhat tedious, but it is doable.

36. Which "and"- and "or"-operations beyond min and max should we choose (cont-d)

- However, expert rules often include propositional connectives like "and" and "or".
- For example, an expert may way: "If the car in front is close *and* it starts braking a little bit, you also need to brake a little bit."
- There are many possible combinations of this type.
- It is not feasible to ask the expert about degree of confidence of all such combinations.
- So, we need to be able:
  - to estimate the expert's degree of confidence in a propositional combination $A \,\&\, B$ and $A \vee B$
  - based on his/her degrees of confidence a and b in the combined statements A and B.

37. Which "and"- and "or"-operations beyond min and max should we choose (cont-d)

- Algorithms for this estimation are known as, corresponding, "and"- and "or"-operations.
- Let us denote them by $f_{\&}(a, b)$ and $f_{\vee}(a, b)$.

38. Which operations should we choose?

- From the commonsense viewpoint, $A \,\&\, B$ and $B \,\&\, A$ mean the same thing.
- It is therefore reasonable to require that for these two combinations, our estimation leads to the same result, i.e., that $f_{\&}(a, b) = f_{\&}(b, a)$.
- In other word, the "and"-operation must be commutative.
- Also, $A \,\&\, (B \,\&\, C)$ and $(A \,\&\, B) \,\&\, C$ mean the same.
- So it is also reasonable to require that for these two combinations, our estimation leads to the same result.
- Thus, we require that $f_{\&}(a, f_{\&}(b, c)) = f_{\&}(f_{\&}(a, b), c)$, i.e., that the "and"-operation must be associative.
- Similar arguments show that the "or"-operation must also be commutative and associative.
- As we have mentioned, in many cases, min and max operations $f_{\&}(a, b) = \min(a, b)$ and $f_{\vee}(a, b) = \max(a, b)$ work well.

39. Which operations should we choose (cont-d)

- However, sometimes, the resulting control is not perfect, and using other "and"- and "or"-operations leads to better results.
- Usually, in such situations, trial-and-error approach is used to select appropriate "and"- and "or"-operations.
- However, usually, computation speed is an important requirement.
- In this case, the arguments that we used in the beginning of this section imply that:
  - these operations should be obtained as compositions of linear combination, min, and max,
  - i.e., that they should be piecewise linear.

40. Which operations should we choose (cont-d)

- Let us consider the simplest case, when:
  - we have only one linear combination $f(a,b) = c_0 + c_a \cdot a + c_b \cdot b$, and
  - min and max are used only to reduce the resulting values to the interval $[0,1]$,
  - i.e., the corresponding "and"- and "or"-operations have the form $\min(1, \max(c_0 + c_a \cdot a + c_b \cdot b, 0))$.
- This reduction:
  - keeps all the values from the interval $[0,1]$ intact,
  - transforms all the values ≤ 0 to 0, and
  - transforms all the values ≥ 1 to 1.
- In this case, commutativity implies that $c_a = c_b$, i.e., that the linear term has the form $c_0 + c_a \cdot a + c_a \cdot b$.

41. Which operations should we choose (cont-d)

- Associativity implies that

$$c_0 + c_a \cdot (c_0 + c_a \cdot a + c_a \cdot b) + c_a \cdot c = a_0 + c_a \cdot a + c_a \cdot (c_0 + c_a \cdot b + c_a \cdot c).$$

- If we open the parentheses, we get:

$$(c_0 + c_a \cdot c_0) + c_a^2 \cdot a + c_a^2 \cdot b + c_a \cdot c = (c_0 + c_a \cdot c_0) + c_a \cdot a + c_a^2 \cdot b + c_a^2 \cdot c.$$

- By comparing coefficients at a in both sides of this equality, we conclude that $c_a^2 = c_a$, thus either $c_a = 0$ or $c_a = 1$.
- If $c_a = 0$, then $f(a, b)$ is simply a constant not depending on a and b at all.
- But we want to make sure that:
  - when a and b correspond to usual truth values 0 and 1,
  - the corresponding operation coincides with the usual boolean operation.

42. Which operations should we choose (cont-d)

- Thus, $c_a = 1$ and thus, the linear form has the form

$$f(a, b) = c_0 + a + b.$$

- For an "and"-operation, we want to have

$$f_{\&}(0, 1) = 0 \text{ and } f_{\&}(1, 1) = 1.$$

- Thus, for the linear form $f(a, b) = c_0 + a + b$, we must have $f(0, 1) \le 0$ and $f(1, 1) \ge 1$.
- Substituting the expression $f(a, b) = c_0 + a + b$ into these inequalities, we conclude that $c_0 + 0 + 1 \le 0$ and $c_0 + 1 + 1 \ge 1$.
- The first inequality implies that $c_0 \le -1$, the second inequality implies that $c_0 \ge 1$.
- Thus, we have $c_0 = -1$.

43. Which operations should we choose (cont-d)

- For this c_0:
  - the largest value of the linear expression,
  - which is attained when both a and b are the largest possible:

$$a = b = 1,$$

  - is equal to $-1 + 1 + 1 = 1$.
- Thus, the linear expression $-1 + a + b$ is always smaller than or equal to 1, and there is no need to cut it from above.
- So, we only need to cut it from below, so we get $f_{\&}(a, b) = \max(a + b - 1, 0)$.
- For an "or"-operation, we want to have $f_{\&}(0, 0) = 0$ and $f_{\&}(0, 1) = 1$.
- Thus, for the linear form $f(a, b) = c_0 + a + b$, we must have $f(0, 0) \le 0$ and $f(0, 1) \ge 1$.

44. Which operations should we choose (cont-d)

- Substituting the expression $f(a,b) = c_0 + a + b$ into these inequalities, we conclude that $c_0 + 0 + 0 \le 0$ and $c_0 + 0 + 1 \ge 1$.
- The first inequality implies that $c_0 \le 0$, the second inequality implies that $c_0 \ge 0$.
- Thus, we have $c_0 = 0$.
- For this c_0, the value of the linear expression $a + b$ is always non-negative.
- So, there is no need to cut it from below.
- So, we only need to cut it from above, so we get

$$f_\vee(a,b) = \min(a + b, 1).$$

45. Which operations should we choose (cont-d)

- Interestingly, these operations $f_{\&}(a,b) = \max(a+b-1,0)$ and $f_{\vee}(a,b) = \min(a+b,1)$ have indeed been successfully used.
- So, the choice of these operations can be also be justified by the same computational complexity arguments.

46. Second idea: symmetries

- Let us show that our second idea – of symmetries:
  - leads to yet another explanation
  - of semi-heuristic ideas like ReLU activation function $s(z) = \max(0, z)$.
- The main idea here is that:
  - while we want to process physical quantities,
  - what we actually process is the *numerical values* of these quantities,
  - and these numerical values change if we change a measuring unit.

47. Second idea: symmetries (cont-d)

- For example, the same height of 1.7 m, when described in centimeters, has a completely different numerical value: 170.
- In general:
  - if we replace the original measuring unit with a new one which is $\lambda > 0$ times smaller,
  - all numerical values multiply by λ: $x \mapsto x' = \lambda \cdot x$.

48. How this idea explains the selection of ReLU

- If we apply the activation function $s(x)$ to the value x in the original measuring units, we get the value $y = s(x)$.
- On the other hand:
  - if we apply the same activation function to the value x' that describes the same quantity in different measuring units,
  - then we get $y' \stackrel{\text{def}}{=} s(x') = s(\lambda \cdot x)$.
- The value y' in the new units corresponds to the value $\lambda^{-1} \cdot y' = \lambda^{-1} \cdot s(\lambda \cdot x)$ in the original units.
- Thus, the use of the original activation function in new units is equivalent – in the original units – to the transformation $x \mapsto \lambda^{-1} \cdot s(\lambda \cdot x)$.
- In other words, it is equivalent to applying the modified activation function $s'(x) \stackrel{\text{def}}{=} \lambda^{-1} \cdot s(\lambda \cdot x)$ in the original units.
- In most physical situations, the selection of the measuring unit is a question of convenience.

49. How this idea explains the selection of ReLU (cont-d)

- There is no physical reasons why, e.g., meters are more adequate than centimeters or inches.
- Thus, the choice of the best activation function should not depend on the choice of the measuring unit.
- We should get the same processing result whatever measuring unit we choose.
- Thus, we must have $s'(x) = s(x)$, i.e. $\lambda^{-1} \cdot s(\lambda \cdot x) = s(x)$.
- If we multiply both sides of this equality by λ, we get $s(\lambda \cdot x) = \lambda \cdot s(x)$.
- For any value $z > 0$, we can take $x = 1$ and $\lambda = z$ and get $s(z) = z \cdot s(1)$, i.e., $s(z) = c_+ \cdot z$, where we denoted $c_+ \stackrel{\text{def}}{=} s(1)$.
- Similarly, for any value $z < 0$, we can take $x = -1$ and $\lambda = -z$ and get $s(z) = -z \cdot -s(1) = z \cdot (-s(-1))$, i.e., $s(z) = c_- \cdot z$, where

$$c_- \stackrel{\text{def}}{=} -s(-1).$$

50. How this idea explains the selection of ReLU (cont-d)

- Of course, we must have $c_- \neq c_+$, because:
  - otherwise, $s(z)$ would be a linear function
  - and the whole purpose of an activation function is to be able to describe non-linear dependencies.
- Once we have ReLU neurons, with activation function $s_0(z) = \max(0, z)$, we can easily implement the above function $s(z)$ as

$$s(z) = c_- \cdot z + (c_+ - c_-) \cdot s_0(z).$$

- Vice versa:
  - if we have neurons with the above piecewise linear activation function $s(z)$,
  - we can easily implement a ReLU transformation as a linear combination of z and $s(z)$.
- Thus, whatever we can compute with ReLU-based neural network we can compute with $s(z)$-based neural network, and vice versa.

51. How this idea explains the selection of ReLU (cont-d)

- From this viewpoint, ReLU and $s(z)$ are equivalent.
- So, a natural symmetry-based requirement implies that we should use neurons which are equivalent to ReLU.
- This provides a second explanation of why ReLU-based neural networks are empirically successful.

52. Acknowledgments

This work was supported in part by:

- National Science Foundation grants 1623190, HRD-1834620, HRD-2034030, and EAR-2225395;
- AT&T Fellowship in Information Technology;
- program of the development of the Scientific-Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478, and
- a grant from the Hungarian National Research, Development and Innovation Office (NRDI).