4-1-2021

Why, in Deep Learning, Non-Smooth Activation Function Works Better Than Smooth Ones

Daniel Cruz
The University of Texas at El Paso, djcruz@miners.utep.edu

Richard Godoy
The University of Texas at El Paso, rgodoy@miners.utep.edu

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

Comments:
Technical Report: UTEP-CS-21-41

Why, in Deep Learning, Non-Smooth Activation Function Works Better Than Smooth Ones

Daniel Cruz, Ricardo Godoy, and Vladik Kreinovich

Abstract Since in the physical world, most dependencies are smooth (differentiable), traditionally, smooth functions were used to approximate these dependencies. In particular, neural networks used smooth activation functions such as the sigmoid function. However, the successes of deep learning showed that in many cases, non-smooth activation functions like $\max(0,z)$ work much better. In this paper, we explain why in many cases, non-smooth approximating functions often work better – even when the approximated dependence is smooth.

1 Formulation of the Problem

The world is mostly smooth. In the physical world, most dependencies are smooth (differentiable) – phase transitions and explosions are a few exceptions; see, e.g., [4, 9].

Because of this, we usually try smooth models. Because most real-life dependencies are smooth, a reasonable idea is to fit data with smooth dependencies.

In particular, this applies to machine learning, especially to neural networks. In a neural network, we intertwine linear combinations

$$y = c_0 + c_1 \cdot x_1 + \ldots + c_n \cdot x_n$$

and non-linear steps, where the input signal z is transformed into an output $s(z)$ for some non-linear functions $s(z)$. This non-linear function is known as the *activation function*.

Daniel Cruz, Ricardo Godoy, and Vladik Kreinovich
Department of Computer Science, University of Texas at El Paso, 500 W. University
El Paso, TX 79968, USA
e-mail: djcruz@miners.utep.edu, rgodoy@miners.utep.edu, vladik@utep.edu

Any linear function is, of course, always differentiable. So, to make sure that everything is differentiable, we need to make sure that the activation function is smooth. This is exactly what researchers did in the neural networks until the last decade, when they used smooth activation functions, e.g., the sigmoid function

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

see, e.g., [1].

Surprisingly, a not-everywhere-smooth function works much better. However, now it turned out that much better results are obtained when we use non-smooth activation functions such as rectified linear function

$$s(z) = \max(0, z)$$

which is not differentiable at the point $z = 0$; see, e.g., [6].

But why? Why are non-smooth functions better – even if we approximate a smooth dependence?

Some explanations for why rectified linear activation function works better are possible; see, e.g., [5, 7]. However, this explanation is purely mathematical, it does not provide a clear explanation of why non-smooth functions work better than smooth ones.

2 Our Explanation

Decision making – the ultimate goal of science and engineering. One of the ultimate goals of all human activities is to make decisions. This is why we predict weather: we want to decide what to wear tomorrow. This is why we study nuclear physics – we want to find new isotopes for medical applications, new ways to generate and store energy, etc.

From this viewpoint, instead of going into technical details and analyzing how a function can be approximated, let us start with this ultimate goal, let us start with decision making.

We will show that already in the simplest case of decision making, we can find that non-smooth approximations are more efficient.

The simplest case of decision making: majority rule. Most decisions affect several people. Therefore, when making a decision, we need to take into account the effect of different possible decisions on different people.

Usually, different people are effected differently: e.g., when we build a plant, people living near this plant are affected much more than people living reasonably far way from this plant. In many real-life decision making, we need to take this difference into account.

Let us consider the simplest case of decision making, when all people are affected equally. In this case, the decision on whether to accept a certain proposal or not is usually decided by the majority rule (also known as voting): if the majority votes for, the proposal is accepted.

Let us describe the majority rule in precise terms. For simplicity, let us assume that no one abstains, that everyone votes yes or no. Let us denote the result of the i-th person's vote by x_i:

- if the i-th person voted "yes", we take $x_i = 1$, and
- if the i-th person voted "no", we take $x_i = -1$.

The majority rule $y = f(x_1, \ldots, x_n)$ means that:

- if most people voted for, then we should take $y = 1$; and
- if most people voted against, then we should take $y = -1$.

This is the function that we want to approximate.

How can we come up with a smooth approximation? The usual way to approximate a dependence by a smooth function is to use the fact that sufficiently smooth functions can be expanded in Taylor series. So, we can take the first few terms in the corresponding series, and use the resulting polynomial sum as an approximation to the desired function.

This is the usual practice in physics, where we first use linear approximation, then – if needed – a quadratic one, etc. [4, 9]. This is how most functions are computed in a computer: e.g., to compute $\exp(x)$, we take into account this function's Taylor expansion

$$\exp(x) = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + \ldots,$$

and use an approximating polynomial

$$\exp(x) \approx 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!}.$$

How many computational steps do we need to compute a polynomial? For a generic linear polynomial

$$f(x_1, \ldots, x_n) = a_0 + \sum_{i=1}^{n} a_i \cdot x_i,$$

we need to compute the sum of n terms, so we need $O(n)$ computational steps.

To compute a generic quadratic polynomial

$$f(x_1, \ldots, x_n) = a_0 + \sum_{i=1}^{n} a_i \cdot x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot x_i \cdot x_j,$$

we need to compute the sum of $O(n^2)$ terms, so we need $O(n^2)$ computational steps.

To compute a generic cubic polynomial

$$f(x_1,\ldots,x_n) = a_0 + \sum_{i=1}^{n} a_i \cdot x_i + \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} \cdot x_i \cdot x_j + \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} a_{ijk} \cdot x_i \cdot x_j \cdot x_k,$$

we need to compute the sum of $O(n^3)$ terms, so we need $O(n^3)$ computational steps.

To compute a generic polynomial of degree d

$$f(x_1,\ldots,x_n) = a_0 + \sum_{i=1}^{n} a_i \cdot x_i + \ldots + \sum_{i_1=1}^{n} \ldots \sum_{i_d=1}^{n} a_{i_1\ldots i_d} \cdot x_{i_1} \cdot \ldots \cdot x_{i_d},$$

we need to compute the sum of $O(n^d)$ terms, so we need $O(n^d)$ computational steps.

How difficult is it to approximate majority rule by a polynomial? It is known [2, 3, 8] that to approximate the majority-rule function $f(x_1,\ldots,x_n)$ by a polynomial, we need a polynomial of degree $d = \Omega(n)$ – i.e., $d \geq c \cdot n$ for some c. This means that we need $O(n^d) = O(n^{\Omega(n)})$, i.e., exponentially many computational steps – which, for large n, is not practically feasible: for large n, we will need more steps than the lifetime of the Universe.

What if we use non-smooth approximating functions? If we allow non-smooth functions like min and max, then we can easily describe the majority rule in a very simple and easy-to-compute, as

$$f(x_1,\ldots,x_n) = \max(-1, \min(x_1 + \ldots + x_n, 1)).$$

Indeed:

- If most people voted "for", this means that we have more positive terms $x_i = 1$ than negative terms $x_i = -1$. Thus, the resulting sum $x_1 + \ldots + x_n$ is positive. Since all the values x_i are integers, their sum is also an integer, so it must be a positive integer. Every positive integer is greater than or equal to 1, so

  $$\min(x_1 + \ldots + x_n, 1) = 1.$$

  Thus,

  $$\max(-1, \min(x_1 + \ldots + x_n, 1)) = \max(-1, 1) = 1,$$

  which is exactly what we want.
- If most people voted "against", this means that we have more negative terms $x_i = -1$ than positive terms $x_i = 1$. Thus, the resulting sum $x_1 + \ldots + x_n$ is negative. Since all the values x_i are integers, their sum is also an integer, so it must be a negative integer. Every negative integer is smaller than or equal to -1, so $\min(x_1 + \ldots + x_n, 1) = x_1 + \ldots + x_n$ and

  $$\max(-1, \min(x_1 + \ldots + x_n, 1)) = \max(-1, x_1 + \ldots + x_n) = -1,$$

which is exactly what we want.

Conclusion. Already in the very simplest case of decision making, the use of non-smooth functions drastically decreases the computation time needed for approximating the desired dependence.

Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science), and HRD-1834620 and HRD-2034030 (CAHSI Includes).

It was also supported by the program of the development of the Scientific-Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478.

References

1. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2006.
2. M. Bun, R. Kothari, and J. Thaler, "Quantum algorithms and approximating polynomials for composed functions with shared inputs", *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms SODA'2019*, San Diego, California, January 6–9, 2019.
3. M. Bun and J. Thaler, "Approximate degree in classical and quantum computing", *ACM SIGACT News*, 2020, Vol. 51, No. 4, pp. 48–72.
4. R. Feynman, R. Leighton, and M. Sands, *The Feynman Lectures on Physics*, Addison Wesley, Boston, Massachusetts, 2005.
5. O. Fuentes, J. Parra, E. Anthony, and V. Kreinovich, "Why rectified linear neurons are efficient: a possible theoretical explanations", In: O. Kosheleva, S. Shary, G. Xiang, and R. Zapatrin (eds.), *Beyond Traditional Probabilistic Data Processing Techniques: Interval, Fuzzy, etc. Methods and Their Applications*, Springer, Cham, Switzerland, 2020, pp. 603–613.
6. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Leaning*, MIT Press, Cambridge, Massachusetts, 2016.
7. V. Kreinovich and O. Kosheleva, "Optimization under uncertainty explains empirical success of deep learning heuristics", In: P. Pardalos, V. Rasskazova, and M. N. Vrahatis (eds.), *Black Box Optimization, Machine Learning and No-Free Lunch Theorems*, Springer, Cham, Switzerland, 2021, pp. 195–220.
8. A. Tal, "Formula lower bounds via the quantum model", *Proceedings of the 2017 ACM Symposium on Theory of Compututing STOC'2017*, Montreal, Quebec, Canada, June 19–23, 2017, pp. 1256–1268.
9. K.S. Thorne and R.D. Blandford, *Modern Classical Physics: Optics, Fluids, Plasmas, Elasticity, Relativity, and Statistical Physics*, Princeton University Press, Princeton, New Jersey, 2017.