7-1-2021

What Fuzzy and Quantum Computing Can Learn from the Success of Deep Learning

Shahnaz Shahbazova
Azerbaijan Technical University, shahbazova@gmail.com

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

Comments:
Technical Report: UTEP-CS-21-66

What Fuzzy and Quantum Computing Can Learn from the Success of Deep Learning

Shahnaz Shahbazova[1] and Vladik Kreinovich[2]
[1]Azerbaijan Technical University
Baku, Azerbaijan
shahbazova@gmail.com
[2]Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
vladik@utep.edu

Abstract

How can we apply the ideas that made deep neural networks successful to other aspects of computing? For this purpose, we reformulate these ideas in a more general form – and we show that this generalization also covers fuzzy and quantum computing. This enables us to suggest that similar ideas can be helpful for fuzzy and quantum computing as well. In this suggestion, we are encouraged by the fact that as we show, to some extent, these ideas are already helpful.

1 Formulation of the Problem

A natural idea is to learn from successes. We all, scientists and practitioners, try our best to solve our problems better – more effectively, more efficiently, etc. We want to better (and faster) predict all aspects of the future state of the world, we want to compute designs and controls that will make the world's future state even better.

From this viewpoint, every time an approach leads to a success, a natural idea is: how can we use the corresponding successful idea(s) to make improvements in other areas as well?

From this viewpoint, what can we learn from the successes of deep learning. At present, in AI – and, arguably, in computer science in general – one of the most successful directions is deep learning; see, e.g., [4]. It is therefore reasonable to ask: how can we use the main ideas behind this success to make improvements in other directions of computing as well?

This is the problem that we concentrate on in this paper.

Comment. Of course, some researchers may say (and have said): just abandon your less-successful ideas and jump on deep learning bandwagon – but, in contrast to these researchers, we believe that all approaches have potential and can benefit each other. And we will provide examples showing that our belief is justified.

Why quantum and fuzzy computing? In other to analyze how we can generalize the success of neural networks – in their deep learning form, a natural idea is to reformulate the main ideas behind neural networks and deep learning ideas in a more general form. This is what we will do, and we will show that this generalization naturally leads to fuzzy and quantum computing.

2 Let Us Reformulate the Main Ideas Behind Neural Networks and Deep Learning in a More General Form

What is the main challenge of computing? In order to come up with the desired generalization, let us recall what is the main problem of computing now.

For many practical problems, we have, at least on the theoretical level, algorithms for the desired prediction and/or for the desired control. For many problems, these algorithms are practically useful. For example, the existing algorithms can predict tomorrow's weather reasonably well – not perfectly well, but definitely much better than it was possible even a few years ago. Many of these algorithms are so good that, e.g., modern airplanes are designed and tested on computer simulations – and the following flight tests only confirm the simulation results. Self-driving cars, while they still have accidents, are already, on average, much safer than human drivers.

However, in some problems, we have algorithms, but these algorithms require so much computation time that they become practically useless. For example, it is possible to predict in which direction a potentially deadly tornado will turn in the next 15 minutes – this can be done by using algorithms similar to weather prediction – but this prediction is practically useless since it takes several hours on a high-performance computer. By the time we have computed this prediction, the tornado has already moved.

Such examples are plentiful. The need to make computations faster is one of the main challenges of computing.

This challenge is objective. In some cases, it is possible to come up with faster algorithms for solving the same problem. However, in general, there is a limit to how much we can gain this way. Many practical problems are known to be NP-hard. This means that unless P = NP (which most computer scientists believe to be impossible), no feasible algorithm can solve all the instances of this general problem; see, e.g., [6, 11]. In other words, no matter how clever our algorithms, there will always be instances on which these algorithms will take too long to be practically useful.

Since there is a limit to how much we can speed up computations by coming up with better algorithms, it is crucially important to make sure that the implementation of current algorithms is as fast as possible.

How to make the algorithm implementation faster? An algorithm, by definition, consists of elementary computational steps. What are these elementary steps – i.e., hardware supported elementary operations – depends on the computational device. So, to speed up computations, we need:

- to select elementary computational steps that can be computed in the fastest possible way, and
- to actually implement these elementary steps in the fastest possible manner.

Let us analyze these two aspects one by one.

Which elementary steps are the fastest? In sensors and in computers, information is transferred by electric signals.

Some signals are analog. In these signals, the information is conveyed by different values of current and/or voltage. This is how most sensors operate: e.g., a photosensor transforms the intensity of light into the intensity of the corresponding current.

For electric signals, all basic transformations are linear: Ohm's law – according to which voltage V linearly depends on current I, as $V = I \cdot R$ – is clearly linear, more general Kirkhoff's laws are also linear. For analog signals, any linear function is easy to implement:

- multiplication by a constant corresponds to using different resistances R, and
- if we bring two currents I_1 and I_2 together, the resulting current I will be equal to their sum $I = I_1 + I_2$.

In principle, we can have non-linear electric devices, but the fastest are linear transformations.

In most modern computational devices, signals are digital, they are represented as a sequence of bits. For numbers represented in binary form, addition or multiplication by a number are still reasonably fast, but they are no longer the fastest possible operations: just like when we add multi-digit numbers, we perform several digit operations, so does the computer. For digital signals, the fewer bits we have to compute, the faster the computations. From this viewpoint, the fastest are operations in which we compute only one bit – i.e., for which, in effect, we decide whether some property is true ot false. For numbers, the only such properties are equality and inequality, so the fastest operations are min and max. Indeed, in each of these operations:

- first, we decide – via computations – which number is smaller (or larger), and

- then return this smaller (or larger) number – without performing any additional computations.

How to actually implement these elementary operations in the fastest possible way? One of the main factors that limits computer speed is the speed of limit – since, according to modern physics, no communication can be faster than the speed of light; see, e.g., [3, 12].

This may sound like a remote restriction, not something to worry about – but we are already close to this limit. For a typical computer which is about 30 cm across, it takes 30 cm / 300 000 km/sec = 1 nanosecond to go across. During this time, the standard 4 Gigahertz computer already performs 4 operations! So, the only way to make computers much faster is to make them much smaller – and this means making all their components much smaller.

Already each memory cell is of the size of thousands or even hundreds of molecules. If we make this cell smaller, its size will approach the size of a single molecule. For such small objects, we can no longer use the laws of Newtonian mechanics, we need to use special physics of microworld knows as quantum physics.

Computing that takes quantum effects into account is known as *quantum computing*; see, e.g., [9]. Interestingly, one of the main features of quantum physics is that most its processes are linear, so we again go to the need to use linear transformations [3, 9, 12].

So which elementary operations are the fastest? Our analysis shows that the fastest possible elementary operations are:

- linear transformations, that transform inputs $x_1, \ldots, x_n$ into their linear combination $w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n$, and
- min- and max-transformations that transform inputs $x_1, \ldots, x_n$ into either $\min(x_1, \ldots, x_n)$ or $\max(x_1, \ldots, x_n)$.

Unfortunately, fastest transformations are not sufficient. It would be great if we could only use these fastest elementary operations, but, unfortunately, they are no sufficient:

- if we only use linear transformations, then we can only compute linear functions – and many real-life processes are non-linear;
- if we only use min and max – both of which select one of the input values as the output – we will always return one of the inputs, and we will never be able to compute anything else.

So, in addition to the fastest elementary operations, we need something else:

- in addition to linear transformations, we need to use some non-linear transformations, and

- in addition to min and max, we need to use some operations that return the value which is different from one of the inputs.

This is exactly what leads us to neural, fuzzy, and quantum computing. Let us first consider the case when we add some non-linear transformation to linear operations. In this cases, computations means that we interchangingly apply linear transformations and some non-linear operations $y = f(x)$. This is exactly what neural networks do. On each layer, each processing unit – called a *neuron* – first computes a linear combination of inputs (or, for next layers, outputs from the previous layer), and then applies some non-linear transformation to the result. For neural networks, the corresponding transformation is known as an *activation function*; see, e.g., [2].

For artificially set up neural networks, we can select any activation function we want. For quantum computing, we have to rely on nature's non-linear process. Such a process is known as *measurement*. If we have a quantum state s which is a linear combination $s = c_1 \cdot s_1 + \ldots + c_n \cdot s_n$ of the basic states $s_1, \ldots, s_n$, then measurement transforms this state into one of the states s_i with probability $|c_i|^2$; see, e.g., [3, 9, **?**].

The operations min and max correspond to the most widely used operations of *fuzzy logic*, corresponding to "and" and "or". In this case, the corresponding additional operation is known as *defuzzification*; see, e.g., [1, 5, 7, 8, 10, 13].

Thus, our general approach to computations indeed leads to neural, fuzzy, and quantum computing.

3 From This General Viewpoint, What Can We Learn from Deep Learning?

A seemingly reasonable idea. At first glance, the situation is straightforward:

- in addition to the fastest elementary computational steps, we need at least one not-so-fast more complex one;
- overall, we want to speed up computations;
- thus, we need to limit the number of not-so-fast computational steps to a minimum – if possible, to just one such step.

As a result, it is reasonable to use exactly one not-so-fast computational step.

This is exactly how traditional neural networks worked. This is exactly how traditional neural networks worked (see, e.g., [2]):

- first, each neuron k transformed the inputs $x_1, \ldots, x_n$ into their linear combination $y_k = w_{k0} + w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n$;
- then, we apply an appropriate nonlinear transformation $z = f(0$ to each of these results, getting $z_k = f(y_k)$ for each k, and

- finally, we apply a linear transformation to the values z_k, resulting in $y = W_0 + W_1 \cdot z_1 + W_2 \cdot z_2 + \ldots$

This is exactly how fuzzy and quantum computing work now. How does a usual fuzzy control works?

- we apply min and max operations to the original values of the measurement functions, and then
- we apply an appropriate defuzzification procedure to the result, generating the recommended control value.

This is also how usual quantum computing algorithms work:

- we perform some linear quantum operations, and then
- we perform a measurement to get the result.

The above seemingly natural idea is only a first approximation. The above idea would work perfectly if computing fastest elementary operation would require no time at all, and the only computing time would be spent of computing other not-so-fast operations. In reality, computing fastest elementary operations also spends time, and these times accumulate. Sometimes, it may be more efficient to perform more not-so-fast operations.

A simply analogy can explain this. Suppose that you own a car. For a car, repairs are usually cheaper than buying a new car. So, by the same logic as we described above, to save money, we should postpone buying a new car until it is absolutely necessary: e.g., when our previous car stops working. This advice is reasonable at the beginning, when the car is not very old, but with time, it becomes more expensive to continuously spend more and more money on more and more frequent repairs than simply to buy a newer car.

Similarly here, while in general, the above logic sounds reasonable in the first approximation, in reality, it may be faster to have two or more not-so-fast elementary operations than to limit ourselves to only one such operation.

When does it make sense to use several not-so-fast operations. When we have a small number of inputs and simple computations, probably the above argument works: not-so-fast elementary operations are still much slower than all corresponding faster elementary operations taken together. In this case, limiting ourselves to only one not-so-fast operation makes perfect sense.

However, as the number of inputs grows and the computations become more complex, the overall time needed for all elementary operations become comparable with – and even larger – than the time needed for a single not-so-fast operation. In this case, using more than one not-so-fast operation may be beneficial.

What deep learning showed. This is exactly what happened in neural networks: it turned out that in many complex applications, it is more beneficial

to use several nonlinear layers. This is what is called *deep learning* – when we have several nonlinear layers.

So what can fuzzy and quantum computing learn from this success. In view of the above analysis, a natural idea is to allow several not-so-fast layers.

In fuzzy control, it means that instead of a single one-stage fuzzy controller, we should use multi-stage (e.g., hierarchical) fuzzy controllers, where results of some stages are used as input to other controllers. This indeed works – hierarchical fuzzy controllers have been shown to be efficient in controlling complex systems.

In quantum computing, it means that instead of trying to fit everything into a single quantum algorithm, it may be better to have a multi-stage algorithms, in which we first perform measurements, and then do something with this measurement results. This also works – e.g., this is how quantum cryptography works: we first perform measurements, and then process the results of these measurements; see, e.g., [9].

Our examples show that this idea is not as new and as revolutionary as it may seem at first glance. Our point is that at present, this idea is not the mainstream neither in fuzzy nor in quantum computing. Our argument is that, as we handle more and more complex problems, with more and more inputs, this idea should become mainstream.

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), and by the AT&T Fellowship in Information Technology.

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] R. Belohlavek, J. W. Dauben, and G. J. Klir, *Fuzzy Logic and Mathematics: A Historical Perspective*, Oxford University Press, New York, 2017.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2006.

[3] R. Feynman, R. Leighton, and M. Sands, *The Feynman Lectures on Physics*, Addison Wesley, Boston, Massachusetts, 2005.

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

[5] G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.

[6] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1998.

[7] J. M. Mendel, *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions*, Springer, Cham, Switzerland, 2017.

[8] H. T. Nguyen, C. L. Walker, and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2019.

[9] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press: Cambridge, U.K., 2000.

[10] V. Novák, I. Perfilieva, and J. Močkoř, *Mathematical Principles of Fuzzy Logic*, Kluwer, Boston, Dordrecht, 1999.

[11] C. H. Papadimitriou, *Computational Complexity*, Addison Wesley, San Diego, CA, 1994.

[12] 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.

[13] L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.