9-1-2021

Localized Learning: A Possible Alternative to Current Deep Learning Techniques

Javier Viana
University of Cincinnati, vianajr@mail.uc.edu

Kelly Cohen
University of Cincinnati, cohenky@ucmail.uc.edu

Anca Ralescu
University of Cincinnati, ralescal@ucmail.uc.edu

Stephan Ralescu
University of Cincinnati, ralescs@mail.uc.edu

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

Comments:
Technical Report: UTEP-CS-21-85

Localized Learning: A Possible Alternative to Current Deep Learning Techniques

Javier Viaña, Kelly Cohen, Anca Ralescu, Stephan Ralescu, and Vladik Kreinovich

Abstract At present, the most efficient deep learning technique is the use of deep neural networks. However, recent empirical results show that in some situations, it is even more efficient to use "localized" learning – i.e., to divide the domain of inputs into sub-domains, learn the desired dependence separately on each sub-domain, and then "smooth" the resulting dependencies into a single algorithm. In this paper, we provide theoretical explanation for these empirical successes.

1 Formulation of the Problem

Deep learning is successful but it is not a panacea: sometimes it cannot be used. In many problems, deep learning techniques – see, e.g., [4] – provide the most efficient learning: they lead to the most accurate approximation to the real-life phenomena.

This does not mean, of course, that no other machine learning techniques are needed: in some other situations, alternative machine learning techniques are needed. For example, it is known that deep learning requires a large amount of data and a lot of computation time. So, if we do not have enough data and/or we do not have enough time to train the network, we have to use other machine learning tools.

Even when deep learning can be used, other methods are sometimes better. Interestingly, it turns out that in some situations, even when there is enough data and enough time to use deep learning, alternative methods still lead to more accurate

Javier Viaña, Kelly Cohen, Anca Ralescu, and Stephan Ralescu
University of Cincinnati, Cincinnati, Ohio 45219, USA
e-mail: vianajr@mail.uc.edu, cohenky@ucmail.uc.edu, ralescal@ucmail.uc.edu, ralescs@mail.uc.edu

Vladik Kreinovich
Department of Computer Science, University of Texas at El Paso, 500 W. University El Paso, Texas 79968, USA, e-mail: vladik@utep.edu

results; see, e.g., [12, 13]. Specifically, these papers use the following "localized" learning idea:

- we divide the area of possible values of the input into several sub-domains,
- we learn the dependence "locally" – i.e., separately on each sub-domain, and then
- we combine the resulting dependencies by making smooth transitions between them – this can be naturally done, e.g., by using fuzzy techniques (see, e.g., [1, 5, 7, 9, 10, 14]).

Comment. Of course, it is important to make sure that the comparison between different techniques is fair. For each machine learning technique, the more parameters we use, the more accurate results we get. So, the only way to claim that one technique is more accurate is:

- either to compare variants of these two methods that use the same number of parameters,
- or, alternatively, to show one of the techniques requires fewer parameters to reach the same approximation accuracy.

The second alternative is exactly how the comparison was performed in [12, 13].

A natural question. Empirically, the results from [12, 13] are interesting, but in view of the current successes of deep learning, these results are somewhat unexpected. So, a question naturally arises:

Why are these localized techniques so much more accurate than deep learning?

What we do in this paper. To answer this question, we first ask a related question:

Why are deep learning techniques so successful?

To answer this auxiliary question, we first go even further and ask:

Why are neural networks so successful in the first place?

Once we answer these two questions and find out what are the strong points of neural networks (including deep ones), it will also become clear what are the limitations of neural networks, and why shallow localized networks can, in some important practical situations, overcome these limitations.

In line with this plan:

- in Section 2, we analyze why neural networks are successful in the first place,
- in Section 3, we focus on successes of deep learning, and
- finally, in Section 4, we provide a possible explanation of why localized methods are sometimes better.

Comment. Most main ideas described in Sections 2 and 3 first appeared in [6]; ideas described in Section 4 are completely new.

2 Why Neural Networks: A Theoretical Explanation

Why do we need a theoretical explanation. Neural networks appeared as a way to simulate how we humans solve problems: in our brains, signals are processed by neurons. The fact that we are the result of billions of years of improving evolution makes researchers believe that many biological processes are optimal (or at least close to optimal), so simulating them makes perfect sense.

On the other hand, there is a difference between computers and us: computers are built from different materials than our brains, they operate on a different size and time scales, so an optimal solution for a computer may be different from the optimal solution for a brain. For example, birds have wings to fly, and airplanes – that, to some extent, simulate the birds – also have wings, but while for the birds, flapping the wings is the optimal flying strategy, airplane wings do not flap at all.

To design an airplane, it is not enough to copy the birds, we also need to perform some theoretical analysis. Similarly, to decide which features should be used in computing, it is desirable to provide a theoretical analysis.

The main objective of the original neural network: computation speed. Artificial neural networks appeared when the computation speed of computers was several orders of magnitude lower than now. This relatively slow speed was the main bottleneck, preventing computers from solving many practical problems. So, the question arose: how can we make computers faster?

Main idea: parallelism. If a person has a task that takes too long – e.g., cleaning several offices, then to speed it up, a natural idea is to ask for help. If several people work on the same task, this task gets performed much faster. Similarly, if it takes too long for one processor to solve a problem, a natural idea is to have many processors working in parallel:

- on the first stage, all the processors perform some tasks,
- after that, on the second stage, processors use the results of the first stage to perform additional computations, etc.

To decrease the overall computation time, we need to minimize the number of stages (also called *layers*), and to make each stage as fast as possible.

Linear vs. nonlinear functions. We consider deterministic computers, where the result is uniquely determined by the inputs. In mathematical terms, this means that on each stage, what each processor computes is a function of the inputs. In these terms, to select fast stages, we need to decide which function are the fastest to compute.

Functions can be linear or nonlinear. Of course, linear functions are easier – and thus, faster – to compute. However, we cannot have processor computing only linear functions – because in this case, all the computer will compute will be compositions of linear functions, and such compositions are themselves linear. On the other hand, many real-life dependencies are nonlinear.

Thus, in addition to processors computing linear functions, we also need processors computing nonlinear functions.

Which non-linear functions are the fastest to compute? In general, the fewer variables the function has, the faster it is to compute. Thus, the fastest to compute are nonlinear functions with the smallest possible number of inputs: namely, a single input.

Resulting computation scheme. We thus arrive at a scheme at which at each stage, processors compute:

- either a linear function $y = a_0 + \sum_{i=1}^{n} a_i \cdot x_i$
- or a nonlinear function of one variable $y = s(x)$; such a function is known as an *activation function*.

From the viewpoint of minimizing computation time, it makes no sense to have two linear stages one after another, since the composition of two linear functions is also linear – so we can replace these two stages by a single stage. Similarly, it makes no sense to have two nonlinear stages one after another, since a composition $s_1(s_2(x))$ of two functions of one variable is also a function of one variable. Thus, linear and nonlinear stages must interleave.

It turns out that both 2-stage schemes:

- a linear (L) stage followed by a nonlinear (NL) stage, i.e., a sequence L–NL, and
- a nonlinear stage followed by a linear stage: NL–L

cannot accurately represent general continuous functions on a bounded domain – neither of them can even represent the function $f(x_1,x_2) = x_1 \cdot x_2$ with sufficient accuracy. However, both 3-stage schemes L–NL–L and NL–L–NL can approximate any function. Since NL takes longer than L, the scheme L–NL–L is clearly the fastest.

In this scheme:

- First, each processor k ($k = 1,\ldots,K$) transforms the inputs x_i into their linear combination $y_k = w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n - w_{k0}$.
- On the next stage, a nonlinear transformation is applied to each y_k, so we compute the values $z_k = s_k(y_k)$.
- Finally, in the final third stage, we compute a linear combination

$$y = W_1 \cdot z_1 + \ldots + W_K \cdot z_K - W_0,$$

i.e.,

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

This is exactly the formula for the traditional 3-layer neural network.

Usually, in this network, all the processors (called *neurons*) use the same activation function: $s_1(x) = \ldots = s_K(x) = s(x)$. Then, the formula (1) takes the following form:

$$y = \sum_{k=1}^{K} W_k \cdot s\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right) - W_0. \tag{2}$$

Comment. In most cases, traditional neural networks use a so-called sigmoid activation function

$$s(x) = \frac{1}{1+\exp(-x)}.$$

3 Need to Go Beyond Traditional Neural Networks and Deep Learning

At present, computation speed is no longer the major concern, accuracy is. While several decades ago, when neural networks were invented, computational speed was the main bottleneck, at present, computers are much faster.

The main concern now is not the speed, but how accurately we can perform the computations – how accurately we can predict weather, how accurately we can estimate the amount of oil in a given oilfield, etc.

How to increase accuracy. The more parameters we use, the better we can fit the data and thus, the more accurate the model.

From this viewpoint, the larger the number K of the neurons, the more parameters we have in the formula (2), and thus, the more accurately we can represent any given function.

Limitation. However, there is a serious limitation in this increase of number of options – caused by the fact that any perturbation of K neurons does not change the expression (2).

There are $K!$ such permutations – which, for large K, is a huge number. So, while we have many possible combinations of the coefficients w_{ki} and W_k, there are much fewer ($K!$ times fewer) different functions represented by these combinations.

How to overcome this limitation. To overcome this limitation, a natural idea is to decrease the number K of neurons in each layer. To preserve the same number of parameters, we therefore need to place some neurons in other layers. Thus, instead of the original 3-layer configuration L–NL–L, we get a multi-layer configuration

L–NL–L–NL–...

This is what a deep neural network is about. In a nutshell, this multi-layer scheme is exactly what is known as a deep neural network.

There are also some other differences: e.g., deep learning mostly uses a different activation function $s(x) = \max(x, 0)$ known as *rectified linear unit* (ReLU, for short).

4 Beyond Deep Learning, Towards Localization

Let us get back to estimating accuracy. As we have mentioned, at present, the main objective in most computational problems is accuracy. In the previous section, we explained how transition to deep neural networks helps increase accuracy. But maybe there are other ways to do it?

To answer this question, let us analyze the problem of decreasing accuracy somewhat more deeply.

How to estimate accuracy with which we know the parameters: general reminder. Suppose that to "train" the system, i.e., to find the values of the corresponding parameters, we can use M measurement results, with an average accuracy ε.

Let us denote by P the overall number of parameters in the model. Each measurement result means one equation in which the parameters are unknowns. So, if we take all measurement results into account, we have M equations with P unknowns. In general, if we have fewer equations than unknowns, then we cannot uniquely determine all the unknowns, some of them may be set arbitrarily – so we do not need all P parameters. Thus, we must have $M \geq P$ – and usually, we have $M \gg P$.

If we had exactly P observations, then we could determine each parameters with accuracy proportional to ε. Since we have a duplication – i.e., we have more observations than unknowns – we can use this duplication to make the results more accurate. In general, according to statistics, if we have d measurements to determine a parameter, the accuracy increases by a factor $\sqrt{d}$; see, e.g., [11].

In our case, we have M measurements for P paremeters, so, on average, we have $d = M/P$ measurements per parameter. Thus, we can determine each parameter with accuracy proportional to

$$\delta = \frac{\varepsilon}{\sqrt{M/P}} = \sqrt{\frac{P}{M}} \cdot \varepsilon.$$

What is the accuracy of the result of using this model. The value δ describes the accuracy with which we know each parameter. These accuracies affect the accuracy of prediction. In general, each predicted values depends on all P parameters. each parameters contributes accuracy $\sim \delta$ to the prediction result. It is reasonable to assume that these P contributions are independent. Thus, according to statistics [11], the overall effect of all these contributions is proportional to $\sqrt{P} \cdot \delta$.

How can we make predictions more accurate. To make the results more accurate, we need to decrease the number of parameters on which each predicted value depends.

If each predicted value depend only on $P' \ll P$ parameters, then the overall inaccuracy of the prediction is proportional to $\sqrt{P'} \cdot \delta$, which is much smaller than $\sqrt{P} \cdot \delta$.

In other words, we need localization. To make sure that $P' \ll P$, we need to make sure that each predicted value depends only on a few of the parameters. Thus, each predicted value $f(x_1, \ldots, x_n)$ depends only a few parameters. In other words, the list

of all P parameters should be divided into sublists (maybe intersecting), so that in each sub-domain of the domain of all possible inputs $x = (x_1, \ldots, x_n)$ we have an expression depending only on a few parameters.

In other words, we have separate few-parametric expressions describing the desired dependence on each sub-domain – this is exactly what is usually mean by *localization* – that the value of the function in each sub-domain is determined only by parameters corresponding to this sub-domain. Such a localization is exactly what is used in [12, 13].

Comments.

- In terms of the function (1), this would mean that instead of using the same activation function $s_k(x) = s(x)$ for all the neurons, as in traditional and in deep neural networks, we use, in effect, different activation functions $s_k(x) \neq s_{k'}(x)$ each of which corresponding to a certain sub-domain of the original domain. When the activation functions are different, there is no $K!$ duplication and thus, no decrease in accuracy – even when we use a traditional ("shallow") scheme.
- There are additional advantages in a localized approach:
  - it is faster to find the parameters: we need to solve a system with fewer unknowns $P' \ll P$, and the computations corresponding to different sub-domains can be performed in parallel, and
  - it is easier to modify the solution when the values change in some sub-domain.
- Similar arguments explain why an approximation by splines – where we have polynomial approximation on each sub-domain and then smooth them out – leads, in general, to a much better accuracy than a "global" (on the whole domain) approximation by a polynomial; see, e.g., [2, 3, 8].

Corollary: shallow or deep. According to the localization idea, each value $f(x)$ depends only on the parameters corresponding to this point x and neighboring points $x' \approx x$. In a multi-layer scheme, this means that:

- the signal produced by the last layer depends only on the values from the previous layer which are close to x;
- these values, in turn, depend only on coefficients of the pre-previous layer which correspond to locations x'' which are close to x', etc.

We have $x' \approx x$, $x'' \approx x'$, etc., i.e., the differences $x' - x$, $x'' - x'$, etc., are small. However:

- the difference $x'' - x$ between x'' and x is the sum of two small differences:

$$x'' - x = (x'' - x') + (x' - x);$$

- if we go back one more layer, then the difference $x''' - x$ is the sum of three small differences, etc.

Thus, the more layers we have, the less localized our system, and therefore, the more parameters we need to take into account to predict each value $f(x)$.

So, to achieve the best accuracy, we need to use the smallest possible number of layers – i.e., one non-linear layer, as in traditional neural networks. This is exactly what is used in [12, 13].

Comment. Interestingly, there is some rudimentary localization effect in deep neural networks as well.

Indeed, since the corresponding activation function $s(x) = \max(x, 0)$ is equal to 0 for half of the inputs – namely, for all negative inputs – this means that, on average, help of the value from the previous layer do not affect the next layer. So, the final value produced by the last layer is determined only by half of the neurons in the previous layer. These values, in turn, depend only on the one half of neurons in the previous layer, etc. So, at least half of the parameters are not used when estimating each value – and this fact decreases the approximation error in comparison with our generic estimates.

5 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);
- 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.

The authors are thankful to all the participants of the International Seminar on Computational Intelligence ISCI'2021 (Tijuana, Mexico, August 17–19, 2021), especially to Oscar Castillo and Patricia Melin, for valuable discussions.

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. de Boor, *A Practical Guide to Splines*, Springer, New York, 2001.
3. P. H. C. Eilers and B. D. Marx, *Practical Smoothing: The Joys of P-splines*, Cambridge University Press, Cambridge, UK, 2021.
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 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.
7. J. M. Mendel, *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions*, Springer, Cham, Switzerland, 2017.

8. G. Micula and S. Micula, *Handbook of Splines*, Springer, Dordrecht, 2008.
9. H. T. Nguyen, C. L. Walker, and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2019.
10. V. Novák, I. Perfilieva, and J. Močkoř, *Mathematical Principles of Fuzzy Logic*, Kluwer, Boston, Dordrecht, 1999.
11. D. J. Sheskin, *Handbook of Parametric and Non-Parametric Statistical Procedures*, Chapman & Hall/CRC, London, UK, 2011.
12. J. Viaña and K. Cohen, "Fuzzy-based, noise-resilient, explainable algorithm for regression", *Proceedings of the 2021 Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'2021*, West Lafayette, Indiana, June 7–9, 2021.
13. J. Viaña, S. Raslecu, K. Cohen, A. Ralescu, and V. Kreinovich, "Extension to multidimensional problems of a fuzzy-based explainable & noise-resilient algorithm", *Proceedings of the 2021 International Workshop on Constraint Programming and Decision Making CoProD'2021*, Szeged, Hungary, September 12, 2021.
14. L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.