Why Deep Neural Networks: Yet Another Explanation

Ricardo Lozano, Ivan Montoya Sanchez, and Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
El Paso, Texas 79968, USA
ralozano3@miners.utep.edu, iamontoyasa@miners.utep.edu,
vladik@utep.edu

Formulation of the problem. A neuron transforms the inputs $x_1, \ldots, x_n$ into a value $y_k(x_1, \ldots, x_n) = s\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right)$ for some coefficients w_{ki}, where $s(z)$ is a nonlinear continuous function known as an *activation function*.

One of the main advantages of biological neural networks is high parallelism: e.g., when we look, millions of neurons simultaneously process different parts of the image. In such heavily parallel system, the processing time is proportional to the number of computational stages. It is known that already for one stage – when signals go into nonlinear neurons, and then a linear combination $y(x_1, \ldots, x_n) = \sum_{k=1}^{K} W_k \cdot y_k(x_1, \ldots, x_n) - W_0$ of neurons' results is returned – has the universal approximation property: for every continuous function $f(x_1, \ldots, x_n)$ on a bounded domain ($|x_i| \le B$ for all i) and for every desired accuracy $\varepsilon > 0$, there exist weights w_{ki} and W_k for which the resulting value $y(x_1, \ldots, x_n)$ is ε-close to $f(x_1, \ldots, x_n)$ for all x_i; [1, 2].

Since one stage is the fastest, and with one stage, we can, in principle, represent any dependence, why do we need multi-stage ("deep") neural networks? But empirically, we do need them: many machine learning problem could not solved with traditional ("shallow") networks, but were successfully solved by deep ones [2]. Why?

A possible explanation. Computational devices are never absolutely accurate – e.g., due to rounding. So, we can only implement an activation function approximately, with some accuracy $\delta > 0$. So, all we know that the actual neurons apply some function $\widetilde{s}(z)$ which is δ-close to $s(z)$.

It is known that every continuous function on a bounded domain can be approximated, with any given accuracy, by a polynomial – this is, by the way, how all special functions like $\exp(x)$, $\sin(x)$, etc. are computed in a computer, by computing the corresponding approximating polynomial. So, it is possible that neurons use a polynomial function $\widetilde{s}(z)$. Let d be the order of this polynomial, i.e., the largest overall degree of its terms. Then, in a traditional neural network, all the outputs $y_k(x_1, \ldots, x_n)$ are polynomials of order d, and thus, their linear combination $y(x_1, \ldots, x_n)$ is also a polynomial of order d – and polynomials of fixed order are *not* universal approximators.

So, we need more than one non-linear layer. What if we have two non-linear layers, so that outputs y_k of the first later serve as inputs to the second one? In this case, the resulting function is obtained by applying a polynomial of degree d to polynomials of degree d – resulting in a polynomial of degree d^2 – thus also not a universal approximator. For any fixed number L of layers, we will get polynomials of degree bounded by d^L.

Thus, in this realistic setting, to get the universal approximation property, we cannot limit the number of layers – and this is what deep networks are about.

References

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

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