Why Sigma-Pi-Sigma Neural Networks Are Efficient: A Mathematical Explanation

Sebastian Marquez[1], Ivan Montoya[1], Randhal S. Ramirez Orozco[1],
Emmanuel Saenz[1], Leobardo Valera[2], and Vladik Kreinovich[1]
[1]Department of Computer Science, University of Texas at El Paso
500 W. University, El Paso, Texas 79968, USA
iamontoyasa@miners.utep.edu, smarquez19@miners.utep.edu,
esaenz16@miners.utep.edu;rsramirezorozc@miners.utep.edu, vladik@utep.edu
[2]Computer Science and Information Technology Systems Department
El Paso Community College, 919 Hunter, El Paso, TX 79915, USA
leobardovalera@gmail.com

What are Sigma-Pi-Sigma Neural Networks? Empirially, the following 3-layer networks work very efficiently [1]: (1) neurons in the first *Sigma* compute linear combinations of the inputs, then (2) neurons in the second *Pi* layer compute products of some results of the first layer, and finally, (3) a linear combination of the 2nd layer results is returned to the user.

A natural question. Why are these neural networks efficient while similar networks with different operations are not as efficient?

These networks compute polynomials. By definition, every function obtained from the variables and constants by addition and multiplication is a polynomial. So, all these networks compute are polynomials of the inputs. Vice versa, each polynomial P can be computed this way: indeed, each polynomial can be represented as a linear combination of monomials, i.e., products of variables. So, to compute P, in the 1st layer we copy each variable, then we compute all monomials in the 2nd layer, and produce the result.

So we can reformulate our question; why polynomials are effective? To be more precise, why out of all possible families of functions $F = \{c_1 \cdot e_1(x_1, \ldots, x_n) + \ldots + c_m \cdot e_m(x_1, \ldots, x_n)\}_{c_1,\ldots,c_m}$, where the smooth functions $e_i(x_1, \ldots, x_n)$ are fixed, polynomials are the most efficient?

Our main idea. We want to process the values of the physical quantities, but what we actually process are their numerical values, and these numerical values depend on the choice of the starting point and of the measuring unit: a good example is Celsius vs Fahrenheit temperature scales. A change in the starting point leads to shift $x \to x + b$, change of the measuring unit leads to scaling $x \to a \cdot x$. It makes sense to require that the results of data processing do not change when we use a different scale. In our case, this means that for each values a_i and b_i, F should be the same as

$$\{c_1 \cdot e_1(a_1 \cdot x_1 + b_1, \ldots, a_n \cdot x_n + b_n) + \ldots + c_m \cdot e_m(a_1 \cdot x_1 + b_1, \ldots, a_n \cdot x_n + b_n)\}_{c_1,\ldots,c_m}.$$

This leads to the desired explanation. It turns out that the above requirement implies that the functions $e_i(x_1, \ldots, x_n)$ are polynomials – and thus, that all the functions from the class F are polynomials [2].

[1] Q. Fan, Q. Huang, J. M. Zurada, J. Peng, and H. Li, "Graph regularization for high order neural networks: boundedness, convergence, and enhanced performance with entropy error function", *IEEE Computational Intelligence Magazine*, May 2026, pp. 84–98.

[2] V. Kreinovich and C. Quintana. "Neural networks: what non- linearity to choose?", *Proceedings of the 4th University of New Brunswick AI Workshop*, Fredericton, N.B., Canada, 1991, pp. 627–637.