Why Residual Neural Networks

Sofia Holguin and Vladik Kreinovich
Department of Computer Science, University of Texas at El Paso
500 W. University, El Paso, TX 79968, USA
seholguin2@miners.utep.edu, vladik@utep.edu

What are neural networks. The basic computations unit of a neural network is a *neuron* that transforms inputs $x_1, \ldots, x_n$ into a value $s(a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n)$ for some constants a_i, where $s(x)$ is a nonlinear function known as an *activation function*. In a neural network, some neurons process the inputs, some process the results of other neurons. Usually, neurons form *layers*: neurons from layer 1 process inputs, neurons of layer 2 process the results of neurons of layer 1, etc. In the last layer, usually, we simply compute a linear combination of the signals from the previous layer.

What are residual neural networks. In the traditional neural networks, the outputs of each layer are the inputs to the next layer. The main idea behind residual neural networks is that each neuron at layer i can use, as inputs, not only the outputs of the previous $(i-1)$-st layer, but also outputs from the layers before it: $(i-2)$-nd, etc.

Residual neural networks are efficient, but why? Empirically, residual neural networks are often more efficient than the traditional ones. In this talk, we provide a possible theoretical explanation for this efficiency.

Our model. In real life applications, most dependencies are smooth. Functions describing many smooth dependencies can be expanded in Taylor series. In this case, the sum of the first few terms in these Taylor series provides a good approximation to the resulting dependence. This is how most special functions like exp, sin, etc. are usually computed.

The simplest nonlinear approximation is when we take into account linear and quadratic terms, i.e., when we consider expressions of the type $f(x_1, \ldots, x_n) = a_0 + \sum\limits_{i=1}^{n} a_i \cdot x_i + \sum\limits_{i,j=1}^{n} a_{ij} \cdot x_i \cdot x_j$. This approximation is what we will consider in our model: both in the description of the function that we want to approximate and in description of the activation function, we will ignore cubic and higher order terms, and assume that all these functions are quadratic.

It is sufficient to consider neurons with activation function $s(x) = x^2$. First, we show that in this approximation, we can replace each neuron by a neuron with $s(x) = x^2$ – at the expense of changing the coefficients in the corresponding linear terms $a_0 + a_1 \cdot x_1 + \ldots$

In this approximation, one nonlinear layer is sufficient. Each term x_i^2 can be obtained by one quadratic neuron, each term $x_i \cdot x_j$ can be obtained as $((x_i + x_j)^2 - (x_i - x_j)^2)/4$, and each term x_i can be obtained as $((x_i+1)^2-(x_i-1)^2)/4$, so one nonlinear layer is sufficient to represent any quadratic expression.

How many neurons we need. Let us denote by k the rank of the matrix a_{ij}. Then, in the new coordinates $z_1, \ldots, z_n$ in which coordinate axes are proportional to eigenvectors, the above quadratic expression takes the form $c_0 + \sum\limits_{i=1}^{n} c_i \cdot z_i + \sum\limits_{i=1}^{k} c_{ii} \cdot z_i^2$. When $k < n$, then:

- traditional neural network needs at least $k+1$ neurons, since otherwise it cannot cover terms proportional to z_{k+1}, but
- with residual neural network, the above formulas enables us to use only k nonlinear neurons.

This explains why residual neural networks are more efficient.