Why Neural Networks Are Computationally Efficient Approximators: An Explanation

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

Abstract

Many real-life dependencies can be reasonably accurately described by linear functions. If we want a more accurate description, we need to take non-linear terms into account. To take nonlinear terms into account, we can either explicitly add quadratic terms to the regression equation, or, alternatively, we can use a neural network with a non-linear activation function. At first glance, regression algorithms would work faster, but in practice, often, a neural network approximation turns out to be a more computationally efficient one. In this paper, we provide a reasonable explanation for this empirical fact.

1 Formulation of the Problem

Practical need to find dependencies. In practice, it often occurs that we know (or conjecture) that a quantity y depends on quantities $x_1, \ldots, x_n$, but we do not know the exact form of this dependence. In such situations, we must experimentally determine this dependence $y = f(x_1, \ldots, x_n)$. For that, in several (S) situations $s = 1, \ldots, N$, we measure the values of both the dependent variable y and of the independent variables x_i. Then, we use the results $\left(x_1^{(s)}, \ldots, x_n^{(s)}, y^{(s)}\right)$ of these measurements to find a function $f(x_1, \ldots, x_n)$ which is consistent with all these measurement results, i.e., for which

$$y^{(s)} \approx f\left(x_1^{(s)}, \ldots, x_n^{(s)}\right)$$

for all s from 1 to S. (The equality is usually approximate since the measurements are approximate and the value y is often only approximately determined by the values of the variables $x_1, \ldots, x_n$.)

First approximation: linear dependence. In many practical situations, the dependence $f(x_1, \ldots, x_n)$ is smooth: informally, this means that small changes in x_i lead to equally small changes in y. In the first approximation, a smooth function can be approximated by its tangent, i.e., by a linear expression

$$f(x_1, \ldots, x_n) = c + \sum_{i=1}^{n} c_i \cdot x_i$$

for appropriate coefficients c and c_i.

The task of estimating the values of these coefficients based on the measurement results $\left(x_1^{(s)}, \ldots, x_n^{(s)}, y^{(s)}\right)$, i.e., based on the system of equations

$$y^{(s)} \approx c + \sum_{i=1}^{n} c_i \cdot x_i^{(s)},$$

is known as *linear regression*; see, e.g., [10].

Need to go beyond linear dependencies. To get a more accurate description of the desired dependence, we need to go beyond the first (linear) approximation.

A natural mathematical approach. A natural mathematical idea – traditionally used in statistical analysis – is that since the first (linear) approximation does not work well, we need to use the second (quadratic) approximation. In other words, we need to describe the desired dependence as

$$f(x_1, \ldots, x_n) = c + \sum_{i=1}^{n} c_i \cdot x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_i \cdot x_j$$

for appropriate coefficients c, c_i, and c_{ij}. Statistical regression methods enable us to find the coefficients from the corresponding system of linear equations:

$$y^{(s)} \approx c + \sum_{i=1}^{n} c_i \cdot x_i^{(s)} + \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_i^{(s)} \cdot x_j^{(s)}.$$

A neural network approach is often more efficient. In practice, often, it is more computationally efficient to use neural networks; see, e.g., [2].

In the traditional (3-layer) neural networks, the input values $x_1, \ldots, x_n$:

- first go through the non-linear layer of "hidden" neurons, resulting in the values

$$y_i = s_0 \left(\sum_{j=1}^{n} w_{ij} \cdot x_j - w_{i0} \right), \quad 1 \le i \le m,$$

- after which a linear neuron combines the results y_i into the output

$$y = \sum_{i=1}^{m} W_i \cdot y_i - W_0.$$

Here, W_i and w_{ij} are *weights* selected based on the data, and $s_0(x)$ is a non-linear *activation function*. Usually, the "sigmoid" activation function is used:

$$s_0(x) = \frac{1}{1 + \exp(-x)}.$$

The weights W_i and w_{ij} are selected so as to fit the data, i.e., that $y^{(s)} \approx f\left(x_1^{(s)}, \ldots, x_n^{(s)}\right)$ for all $s = 1, \ldots, S$.

A natural question. A natural question is: why are neural networks a more computationally efficient approximation? In this paper, we provide an explanation for this empirical phenomenon.

2 Towards an Explanation

Apolloni's idea. One of the problems with the traditional neural networks is that in the process of learning – i.e., in the process of adjusting the values of the weights to fit the data – some of the neurons are duplicated, i.e., we get $w_{ij} = w_{i'j}$ for some $i \neq i'$ and thus, $y_i = y_{i'}$.

As a result, we do not fully use the learning capacity of a neural network, since when $y_i = y_{i'}$, we can get the same approximation with fewer hidden neurons.

To avoid the above redundancy problem, B. Apolloni and others suggested [1] that we *orthogonalize* the neurons during training, e.g., that we make sure that the corresponding linear combinations $\sum_{j=1}^{n} w_{ij} \cdot x_j$ remain orthogonal in the sense that

$$\langle w_i, w_{i'} \rangle \stackrel{\text{def}}{=} \sum_{j=1}^{n} w_{ij} \cdot w_{i'j} = 0$$

for all $i \neq i'$. where we denoted $w_i = (w_{i1}, \ldots, w_{in})$; see also [7, 8].

Neural networks in the second approximation: analysis. We consider the second approximation, in which each function is approximated by a quadratic expression – e.g., by the sum of the constant, linear, and quadratic terms of its Taylor expansion, so that cubic and higher orders can be safely ignored.

In the second approximation, we can approximate the non-linear activation function $s_0(x)$ by the sum of its constant, linear, and quadratic terms:

$$s_0(x) \approx s + s_1 \cdot x + s_2 \cdot x^2.$$

In this case, the above formula for the output of an intermediate neuron takes the following form:

$$y_i = s_0 + s_1 \cdot \left(\sum_{j=1}^{n} w_{ij} \cdot x_j - w_{i0} \right) + s_2 \cdot \left(\sum_{j=1}^{n} w_{ij} \cdot x_j - w_{i0} \right)^2 .$$

The quadratic term in this expression can be described as

$$\left(\sum_{j=1}^{n} w_{ij} \cdot x_j - w_{i0} \right)^2 =$$

$$\left(\sum_{j=1}^{n} w_{ij} \cdot x_j \right)^2 - 2w_{i0} \cdot \left(\sum_{j=1}^{n} w_{ij} \cdot x_j \right) + w_{i0}^2.$$

Here, the term

$$\left(\sum_{j=1}^{n} w_{ij} \cdot x_j \right)^2 = \left(\langle w_i, x \rangle \right)^2$$

is the only quadratic terms, the other terms are linear, where we denote $x = (x_1, \ldots, x_n)$. Thus, the output $y = \sum\limits_{i=1}^{m} W_i \cdot y_i - W_0$ of the neural networks consists of a linear part plus a quadratic part of the type

$$Q_n = \sum_{i=1}^{n} W_i \cdot \langle w_i, x \rangle^2.$$

This part corresponds to the quadratic part $\sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} c_{ij} \cdot x_i \cdot x_j$ of the original Taylor-series representation:

$$Q_n = \sum_{i=1}^{n} W_i \cdot \langle w_i, x \rangle^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_i \cdot x_j.$$

As we have mentioned, it is reasonable to select the vectors w_i to be orthogonal. By dividing each vector by its length (and appropriately multiplying W_i by this length), we can assume that the vectors are also orthonormal, i.e., that $\langle w_i, w_i \rangle = 1$ for all i. In the orthonomal basis formed by these vectors w_i,

- the corresponding matrix c_{ij} becomes a diagonal matrix,
- with values W_i on the diagonal.

Thus:

- the vectors w_i are eigenvectors of the matrix c_{ij}, while
- the values W_i are the eigenvalues of this matrix.

So, we arrive at the following conclusion.

Difference between traditional statistical representation and neural network representation reformulated. In the second approximation, a generic non-linear part of a function can be represented by a general symmetric matrix c_{ij}. We consider the two competing representations of a function $f(x_1, \ldots, x_n)$:

- the traditional statistical representation in terms of the first few terms of Taylor series and
- a neural network representation.

In terms of the matrix c_{ij}, these two representations correspond to the following:

- in the traditional statistical representation, we store all the components c_{ij} of the original matrix;
- in the neural network representation, we store instead the eigenvectors and eigenvalues of this matrix.

Physical analogy: a comment. The above conclusion prompts a natural analogy with quantum physics; see, e.g., [4]. In quantum physics, from the mathematical viewpoint, an observable quantity can be described by a corresponding matrix c_{ij}. However, a more physically natural description is to describe possible values of this quantity – which are exactly eigenvalues of this matrix – and states in which this quantity has these exactly values, which are eigenvectors of the matrix. In this example, a representation via eigenvalues and eigenvectors is clearly intuitively preferable.

Towards efficient computations. Our objective is to come up with an expression that, given the inputs $x_1, \ldots, x_n$, would generate the value $y = f(x_1, \ldots, x_n)$.

Which operations are the most efficient on modern computers? In numerical computations that form the bulk of modern high performance computer usage, the most time-consuming operation is the *dot product*, i.e., computing the $\langle a, b \rangle$ for given vectors a and b.

The prevalence of dot product makes sense from the mathematical viewpoint, since most numerical methods are based on linearization, and in the linear approximation, any function of n variables is approximated as $c + \sum_{i=1}^{n} c_i \cdot x_i$, i.e., as a constant plus a dot product between the vector of inputs and the vector of coefficients.

Not surprisingly, most computer speed-up innovations are aimed at computing the dot product faster – e.g., the multiply-accumulate operation which is an important part of digital signal processing or fused multiple-add operation which is now hardware supported on many modern computers; see, e.g., [3].

From this viewpoint, the way to speed up any computation is to reduce it to as few dot products as possible.

How to efficiently compute $f(x_1, \ldots, x_n)$ under both representations. Computing the value of the linear part requires computing exactly one dot product.

Computing the value of the traditional quadratic form requires $n + 1$ dot products:

- first, we compute n dot products $c_i \stackrel{\text{def}}{=} \sum_{j=1}^{n} c_{ij} \cdot x_j$ for $i = 1, \ldots, n$;
- then, to find the desired value of the quadratic form, we compute the dot product $\sum_{i=1}^{n} c_i \cdot x_i$.

In the neural network representation, to compute the value with a certain accuracy, we can dismiss the terms corresponding to small eigenvalues W_i. As a result, instead of the original formula with n eigenvalues, we get a simplified formula with $n' < n$ eigenvalues:

$$Q_n \approx \sum_{i=1}^{n'} W_i \langle w_i, x \rangle^2.$$

From this representation, we can see that fewer than $n + 1$ dot products are needed:

- first, we compute $n' < n$ dot products $z_i = \langle w_i, x \rangle$ corresponding to n' non-dismissed eigenvectors w_i;
- then, we perform a component-wise vector operation to compute the values $t_i = z_i \cdot z_i$; such vector operations are highly parallelizable and can be performed really fast on most modern computers; see, e.g., [3];
- finally, to find the desired result, we compute the dot product $\sum_{i=1}^{n'} W_i \cdot t_i$.

Resulting comparison. If n' is smaller than n, then indeed the neural network representation can lead to faster computations. This explains the empirical fact that in data processing, neural networks are often more efficient than more traditional statistical methods.

Comment. To "flesh out" this conclusion, we need to estimate to what extent the number n' of non-dismissed eigenvalues is smaller than the number n of all eigenvalues. This estimation is done in the Appendix.

Acknowledgments.

This work was supported in part by the National Science Foundation grants HRD-0734825 and DUE-0926721, by Grant 1 T36 GM078000-01 from the National Institutes of Health, by Grant MSM 6198898701 from MŠMT of Czech

Republic, and by Grant 5015 "Application of fuzzy logic with operators in the knowledge based systems" from the Science and Technology Centre in Ukraine (STCU), funded by European Union.

References

[1] B. Apolloni, S. Bassis, and L. Valerio, "A moving agent metaphor to model some motions of the brain actors", *Abstracts of the Conference "Evolution in Communication and Neural Processing from First Organisms and Plants to Man ... and Beyond"*, Modena, Italy, November 18–19, 2010, p. 17.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2007.

[3] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Francisco, California, 2007.

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

[5] F. Götze and A. Tikhomirov, "Rate of convergence in probability to the Marchenko-Pastur law", *Bernoulli*, 2004, Vol. 10, No. 3, pp. 503–548.

[6] V. A. Marchenko and L. A. Pastur, "Distribution of eigenvalues for some sets of random matrices", *Matematicheski Sbornik*, 1967, Vol. 72(114), No. 4, pp. 507–536.

[7] J. Nava and V. Kreinovich, "Orthogonal bases are the best: a theorem justifying Bruno Apolloni's heuristic neural network idea", *Abstracts of the 9th Joint NMSU/UTEP Workshop on Mathematics, Computer Science, and Computational Sciences*, Las Cruces, New Mexico, April 2, 2011.

[8] J. Nava and V. Kreinovich, "Orthogonal bases are the best: a theorem justifying Bruno Apolloni's heuristic neural network idea", *Journal of Uncertain Systems*, 2012, Vol. 6, to appear.

[9] A. Nica and R. Speicher, *Lectures on the Combinatorics of Free Probability Theory*, Cambridge Univ. Press, Cambridge, UK, 2006.

[10] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, Chapman & Hall/CRC, Boca Raton, Florida, 2007.

A Number of Dismissed Eigenvalues: Semi-Heuristic Statistical Analysis

Idea. The idea is to dismiss some eigenvalues because their contribution is small. Of course, the number of small eigenvalues depends on the matrix c_{ij}.

We would like to know how many such eigenvalues are there *on average*. To formulate this question in precise terms, we need to describe a reasonable probability distribution on the set of all possible matrices.

Random matrices: motivation. In general, for each element c_{ij} of the matrix, we can have both positive and negative values. There are no reasons to expect positive values to be more probable than the negative ones or vice versa. In other words, the situation seems to be symmetric with respect to changing the sign. Thus, the expected value of the element c_{ij} should also be invariant with respect to this transformation. The only number that remains invariant when we change the sign is zero, so we conclude that the mean value of each component c_{ij} should be zero.

Similarly, there is no reason to assume that some of the elements have a different probability distribution; thus, we assume that they are identically distributed. Finally, there is no reason to assume that there is correlation between different elements. Thus, we assume that all the elements are independent. Thus, we arrive at the model in which all the elements are independent identically distributed random variables with mean 0 and a variance σ^2.

Eigenvalues of random matrices. For such random matrices, the distribution of their eigenvalues follows the *Marchenko-Pastur law*; see, e.g., [5, 6, 9]. To be more precise, this law describes the limit case of the following situation. We have an $m \times n$ random matrix X whose elements are independent identically distributed random variables with mean 0 and variance σ^2. Assume that m and n increase in such a way that the ratio m/n tends to a limit $\alpha > 0$. Then, for large n and m, the probability distribution of the eigenvalues of the matrix $Y = XX^T$ is asymptotically equivalent to

$$\rho(x) = \left(1 - \frac{1}{\alpha}\right) \cdot \delta(x) + \rho_c(x),$$

where $\delta(x)$ is Dirac's delta-function (i.e., the probability distribution which is located at the point 0 with probability 1), and $\rho_c(x)$ is different from 0 for $x \in [\alpha_-, \alpha_+]$, where $\alpha_\pm = \sigma^2 \cdot (1 \pm \sqrt{\alpha})^2$, and

$$\rho_c(x) = \frac{1}{2 \cdot \pi \cdot \sigma^2} \cdot \frac{\sqrt{(\alpha_+ - x) \cdot (x - \alpha_-)}}{\alpha \cdot x}.$$

In our case, matrices are square, so $m = n$, $\alpha = 1$ and thus, we have $\alpha_- = 0$, $\alpha_+ = 4\sigma^2$ and thus, the limit probability distribution takes the simplified form

$$\rho(x) = \frac{1}{2 \cdot \pi \cdot \sigma^2} \cdot \frac{\sqrt{(4\sigma^2 - x) \cdot x}}{x}.$$

Eigenvalues x of the matrix $Y = XX^T$ are squares of eigenvalues λ of the original matrix X: $x = \lambda^2$.

We are interested in small eigenvalues. For small eigenvalues, we have $x \ll \sigma$, so the above formula can be further simplified, into

$$\rho(x) \sim \frac{1}{2 \cdot \pi \cdot \sigma^2} \cdot \frac{\sqrt{4\sigma^2 \cdot x}}{x} = \frac{1}{2 \cdot \pi \cdot \sigma^2} \cdot \frac{2 \cdot \sigma \cdot \sqrt{x}}{x} = \frac{1}{\pi \cdot \sigma} \cdot \frac{1}{\sqrt{x}}.$$

The probability density ρ_λ for $\lambda = \sqrt{x}$ can thus be found as

$$\rho_\lambda = \frac{dp}{d\lambda} = \frac{dp}{dx} \cdot \frac{dx}{d\lambda}.$$

For $x = \lambda^2$, we get

$$\frac{dx}{d\lambda} = \frac{d(\lambda^2)}{d\lambda} = 2\lambda,$$

thus,

$$\rho_\lambda(\lambda) = \frac{1}{\pi \cdot \sigma} \cdot \frac{1}{\sqrt{x}} \cdot 2\lambda = \frac{1}{\pi \cdot \sigma} \cdot \frac{1}{\lambda} \cdot 2\lambda = \frac{2}{\pi \cdot \sigma}.$$

This expression for the probability density does not depend on λ at all. Thus, small eigenvalues have an approximately uniform distribution.

Heuristic derivation of the number of eigenvalues that can be safely ignored. We would like to dismiss all the eigenvalues $\lambda_i = W_i$ whose absolute values are smaller than (or equal to) some small number $\delta > 0$. The overall contribution c of these eigenvalues is equal to

$$c = \sum_{i:|\lambda_i| \le \delta} W_i \cdot \langle w_i, x \rangle^2.$$

Since eigenvectors are orthonormal, the n values $\langle w_i, x \rangle^2$ add up to $\langle x, x \rangle^2$. In particular, for unit vectors x, these n values add up to 1. It is reasonable to assume that values corresponding to different eigenvalues are similarly distributed. Under this assumption, all these values have the same mean. The sum of n such means is equal to 1, so each mean is equal to $1/n$.

Each value W_i can be positive or negative. It is reasonable to assume that both negative and positive values are equally possible, so the mean value of each product $W_i \cdot \langle w_i, x \rangle^2$ is 0. Thus, the mean value of the sum is also 0.

Since $\langle w_i, x \rangle^2 \approx \frac{1}{n}$, the variance should be approximately equal to $W_i^2 \cdot \frac{1}{n^2}$. It is also reasonable to assume that the products $W_i \cdot \langle w_i, x \rangle^2$ corresponding to different eigenvalues are independent. Thus, the variance V_c of their sum c is equal to sum of their variances, i.e., to

$$V_c = \frac{1}{n^2} \cdot \sum_{i:|\lambda_i| \le \delta} W_i^2.$$

Since the mean is 0, and c is the sum of the large number of small independent components, it is reasonable to conclude, due to the Central Limit theorem, that

it is approximately normally distributed; see, e.g., [10]. So, with probability 99.9%, all the values of this sum are located within the three sigma interval $[-3\sqrt{V_c}, 3\sqrt{V_c}]$.

Thus, the square root $\sqrt{V_c}$ is a good indication of the size of the dismissed terms. The size of the function itself can be similarly estimates as $\sqrt{V}$, where

$$V = \frac{1}{n^2} \cdot \sum_i W_i^2,$$

and the sum is taken over all eigenvalues. We want to make sure that the dismissed part does not exceed a given portion ε of the overall sum, i.e., that $\sqrt{V_c} \cdot \varepsilon \cdot \sqrt{V}$, or, equivalently, $V_c \le \varepsilon^2 \cdot V^2$.

Within this constraint, we want to dismiss as many eigenvalues as possible; thus, we should not have $V_c \ll \varepsilon^2 \cdot V^2$, because then, we would be able to dismiss more terms. We should thus have $V_c \approx \varepsilon^2 \cdot V^2$. Because of the above expressions for V_c and for V, we therefore get an equivalent formula

$$\frac{1}{n^2} \cdot \sum_{i:|\lambda_i| \le \delta} W_i^2 \approx \varepsilon^2 \cdot \frac{1}{n^2} \cdot \sum_i W_i^2.$$

Multiplying both sides by n^2, we can simplify this requirement into

$$\sum_{i:|\lambda_i| \le \delta} W_i^2 \approx \varepsilon^2 \cdot \sum_i W_i^2.$$

Since the probability distribution of eigenvalues is described by the density function ρ_λ, and the total number of these eigenvalues is n, we have

$$\sum_i W_i^2 \approx n \cdot \int_{-\infty}^{\infty} \lambda^2 \cdot \rho_\lambda(\lambda)\, d\lambda$$

and similarly,

$$\sum_{i:|\lambda_i| \le \delta} W_i^2 \approx n \cdot \int_{-\delta}^{\delta} \lambda^2 \cdot \rho_\lambda(\lambda)\, d\lambda.$$

Thus, the above requirement takes the form

$$n \cdot \int_{-\delta}^{\delta} \lambda^2 \cdot \rho_\lambda(\lambda)\, d\lambda \approx \varepsilon^2 \cdot n \cdot \int_{-\infty}^{\infty} \lambda^2 \cdot \rho_\lambda(\lambda)\, d\lambda.$$

Dividing both sides by n, we can simplify this into

$$\int_{-\delta}^{\delta} \lambda^2 \cdot \rho_\lambda(\lambda)\, d\lambda \approx \varepsilon^2 \cdot \int_{-\infty}^{\infty} \lambda^2 \cdot \rho_\lambda(\lambda)\, d\lambda.$$

For small λ, as we have derived, $\rho_\lambda \approx \text{const}$, so

$$\int_{-\delta}^{\delta} \lambda^2 \cdot \rho_\lambda(\lambda)\, d\lambda \approx \int_{-\delta}^{\delta} \lambda^2 \cdot \text{const}\, d\lambda = \text{const} \cdot \delta^3$$

(for a slightly different constant, of course).

Thus, the above requirement takes the form $\delta^3 \approx \text{const} \cdot \varepsilon^2$, i.e., $\delta \approx \varepsilon^{2/3}$.

Numerical example. So, for example, for $\varepsilon \approx 10\% = 0.1$, we get $\delta \approx 0.1^{2/3} \approx 0.2$, so $\approx 20\%$ of all the eigenvalues can be safely ignored. As a result, we get a 20% decrease in computation time.