# Foundations of Neural Networks Explain the Empirical Success of the "Surrogate" Approach to Ordinal Regression – and Recommend What Next

**Salvador Robles Herrera** and **Martine Ceberio** and **Vladik Kreinovich**
Department of Computer Science, University of Texas at El Paso
500 W. University, El Paso, TX 79968, USA
sroblesher1@miners.utep.edu,mceberio@utep.edu,vladik@utep.edu

## Abstract

Recently, a new efficient semi-heuristic statistical method – called Surrogate Approach – has been proposed for dealing with regression problems. How can we explain this empirical success? And since this method is only an approximation to reality, what can we recommend if there is a need for a more accurate approximation? In this paper, we show that this empirical success can be explained by the same arguments that explain the empirical success of neural networks – and these arguments can also provide us with possible more general techniques (that will hopefully lead to more accurate approximation to real-life phenomena).

**Keywords:** Neural networks, Surrogate Approach, Machine Learning

## 1 Introduction

**Need for regression: a brief reminder.** In real life, in many processes, many quantities are inter-related. Often, we know that a quantity $y$ is largely determined by the value of some other quantity $x$, but we do not know the exact form of this dependence. In other cases, we know that the value of the quantity $y$ is largely determined by the values of several quantities $x = (x_1, \ldots, x_n)$. In such situations, we need to determine this dependence $y = f(x)$ based on the empirical data, i.e., based on several ($K$) cases in which we know both the value $x^{(k)}$ of the quantity $x$ and the value $y^{(k)}$ of the quantity $y$.

In statistics, the problem of determining $f(x)$ from the empirical data is known as *regression*. In computer science, this problem is known as *machine learning*.

**Statistical approach to regression.** While the value $y$ is *largely* determined by the quantity $x$, it is usually not *uniquely* determined by $x$ – there are usually other factors that affect the value $y$. In other words, in different situations in which the value $x$ is the same, we may have different values of $y$. Once we know $x$, we cannot uniquely predict the value $y$: we may have different values $y$ with different probability. At best, based on the value $x$, we can predict the corresponding probability distribution on the set of possible $y$'s – e.g., the probability density $F_x(y)$.

If we know this probability distribution and we want to select a single "most probable" value $y$, a natural idea is to select the value $y$ for which the probability is the largest, i.e., for which the value $F_x(y)$ is the largest.

**Linear regression, logit-type regression, and surrogate approach.** In many practical situations, the dependence of $y$ on $x$ can be well described by a linear function, i.e., by an expression

$$f(x) \approx a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n.$$

One of the main reasons why linear dependencies are ubiquitous is that real-world dependencies are usually smooth (differentiable). Differentiable means that in a small vicinity of each point, the graph of the function is close to its tangent – i.e., to the graph of a linear function. Thus, if the range of $x$ is not too wide, the values $f(x)$ on this range are reasonably close to a linear expression; see, e.g., [2, 8].

So, if we know the probability density function $F(\Delta y)$ that describes the approximation error

$$\Delta y \stackrel{\text{def}}{=} y - (a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n),$$

then for each $x$, the probability distribution of $y$ has the form

$$F_x(y) = F(y - (a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n)).$$

This approach is known as *linear regression*.

In some cases, linear regression does not work well. In such situations, practitioners often use a (generalized version of) so-called *logit regression*, in which the probability distribution of $y$ has the form

$$F_x(y) = F(g(y) - (a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n))$$

for some function $g(y)$.

In some cases, we only have an ordinal scale for the quantity $y$, i.e., we have finitely many possible values of $y$, and all we can say is that some of these values are larger than others. In this case, we can assign numerical values but there are many values to do it: e.g., take the smallest value as 0, the next smallest as 1, etc., or we can use the values $2^0$, $2^1$, etc. To deal with such *ordinal regression* cases, recently, a new technique was proposed – called *surrogate approach* – in which the probability distribution of $y$ has the form

$$F_x(y) = F(g(y) - h(a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n))$$

for some functions $g(y)$ and $h(x)$; see, e.g., [6, 7]. It turns out that this method describes the actual probabilities very well.

**Natural questions.** The first natural question is: why does the surrogate approach work well? Of course, the surrogate approach is not an exact description of reality: it is a good approximation. So, a natural next question is: what model shall we use if we want a more accurate description of the corresponding probabilities?

In this paper, we show that ideas from the foundations of neural networks can provide reasonable answers to these two questions.

## 2 Why Surrogate Approach Is Successful: Possible Statistical Explanation and Its Limitations

**Towards a statistical explanation.** Since we are talking about statistical techniques, let us first look for statistical explanations for the empirical success of the surrogate approach.

As we have mentioned, in most practical situations, the value $y$ is not uniquely determined by the quantities $x_1, \ldots, x_n$, there are other factors – that we will denote by $x_{n+1}$, $x_{n+2}$, ... – about which we have no information but that affect the value $y$.In other words, we have

$$y = F(x_1, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots).$$

For small deviations, as we have mentioned, we can approximate this dependence by a linear expression

$$y \approx a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n +$$
$$a_{n+1} \cdot x_{n+1} + a_{n+1} \cdot x_{n+2} + \ldots$$

We do not know the values of the quantities $x_{n+1}$, $x_{n+2}$, ... So, from our viewpoint, the part

$$\Delta y \stackrel{\text{def}}{=} a_{n+1} \cdot x_{n+1} + a_{n+1} \cdot x_{n+2} + \ldots$$

is unpredictable – i.e., random. In other words, the dependence on $y$ on $x_1, \ldots, x_n$ has the form

$$y = a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n + \Delta y$$

for a random variable

$$\Delta y = y - (a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n).$$

As we have mentioned, if we know the probability density $F(\Delta y)$ for this random variable, then for each values $x_1, \ldots, x_n$, the probability density of $y$ has the form

$$F(y - (a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n)).$$

This is the expression corresponding to the usual linear regression.

In general, we may have different scales for measuring $y$. For example, we can measure the energy of an earthquake in Joules, but it often more convenient to use a logarithmic scale – known as Richter scale. In general, instead of the original value $Y$, we can use a re-scaled value $y = G(Y)$, for some – generally, non-linear – function $G(y)$. In this case, if we know the value $y$ in the new scale, then we can reconstruct the original value $Y$ by applying the inverse transformation $Y = g(y)$, where $g \stackrel{\text{def}}{=} G^{-1}$.

If in the original scale, the value $Y$ was approximately equal to the linear combination of $x_i$, then, for each combination of values $x_1, \ldots, x_n$, the probability distribution for $Y$ is described by the following formula:

$$F(Y - (a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n)).$$

Substituting $Y = g(y)$ into this formula, we get the expression

$$F(g(y) - (a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n))$$

corresponding to the generalized logit regression. In particular, in the 1D case $n = 1$, we get the expression

$$F(g(y) - (a_0 + a_1 \cdot x)).$$

In addition to different scales for $y$, we can have different scales for measuring $x$. In general, instead of the original value $X$, we can use a re-scaled value $x = H(X)$, for some – generally, non-linear – function $H(x)$. In this case, if we know the value $x$ in the new scale, then we can reconstruct the original value $X$ by

applying the inverse transformation $X = h_0(x)$, where $h_0 \stackrel{\text{def}}{=} H^{-1}$. Substituting $X = h_0(x)$ into the formula $F(g(y) - (a_0 + a_1 \cdot x))$, we get

$$F(g(y) - (a_0 + a_1 \cdot h_0(x)),$$

i.e., $F(g(y) - h(x))$, where

$$h(x) \stackrel{\text{def}}{=} a_0 + a_1 \cdot h_0(x).$$

This is exactly the surrogate approach.

**Limitations of the statistical explanation.** The above statistical ideas:

- only explain the surrogate approach for the case of a single variable, and
- do not provide any recommendations about what to do if we want a more accurate description.

To overcome these limitations, we provide a new explanation, an explanation that is based on the ideas underlying neural networks.

## 3 Our Explanation and the Resulting Recommendations

**Why neural networks: a reminder.** In order to come up with our explanation, let us recall the foundations of neural networks.

One of the main advantages of neural networks is that, once trained, they compute the results really fast. This computation speed is a feature that artificial neural networks share with biological neural networks: the reaction time of each neuron is in dozens of milliseconds, but since we have billions of neurons working in parallel, we can solve many problems – such as pattern recognition – at the same speed as computers for which each unit has nanosecond reaction time.

Not only neural networks are fast, but one can show that the desire to make computations fast naturally leads to neural networks; see, e.g., [5]. This argument is as follows. We want a computational device that consists of many computational units working in parallel. To speed up computations, we need to make these units as fast as possible, and we need to have as few sequential layers of these units as possible.

We want a deterministic computer, in which both the intermediate results and the final result of the computation are uniquely determined by the inputs. This means that for each computational unit, the result is uniquely determined by its inputs. In mathematical terms, this means that the output of the computational unit is a function of its inputs. So, to decide which computational units are the fastest, we need to understand which functions are the fastest to compute.

In general, functions can be linear or nonlinear. Clearly, linear functions are easier to compute – and thus, faster to compute. So, we definitely need to have computational units that compute linear functions. However, if we only have linear computational units, we will only be able to compute compositions of linear functions – and such compositions are also linear. But in practice, there are many nonlinear dependencies. Thus, if we want to describe all dependencies in the real world, then, in addition to linear computational units, we also need to have nonlinear units.

Which nonlinear functions are the easiest to compute? In general, the more inputs the function has, the longer it takes to compute the value of this function. Thus, if we want to restrict ourselves to the fastest-to-compute functions, we should thus consider only functions with the smallest possible number of inputs – i.e., functions of one variable.

So, we arrive at a scheme at which on each computational layer, we compute either a linear combination of inputs, or a function $s(x)$ of one variable.

- Since we want the fastest computations, it does not make sense to have two consequent layers in which we compute linear combinations – since a linear combination of linear combinations is also a linear combination of the original inputs, these two layers can be replaced by a single linear layer.
- Similarly, it does not make sense to have two consequent layers in which we compute a function of one variables – since if we first compute $y = f(x)$ and then $z = g(y)$, this is equivalent to compute a single function of one variable $z = g(f(x))$, so these two layers can also be replaced by a single layer.

So, after a linear layer, we must have a nonlinear layer. In each such two-layer fragment, first, a linear layer computes a linear combination

$$y = a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n,$$

and then a nonlinear layer applied some function $z = s(y)$ of one variable to this value $y$, resulting in the value

$$z = s(a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n).$$

This is exactly the usual formula for data processing performed by a neuron [1, 3].

**General idea behind our explanation.** In general, we want to process large amounts of data in reasonable

time. Thus, we want algorithms for processing data to be as fast as possible. Many of these algorithms select the most probable model, i.e.., the model for which the value of the probability density is the largest. To find such a model, we often need to compute the value of the probability density for different models. Thus, this computation should be as fast as possible.

In view of the above, this means that we need to have expressions for this probability distribution in terms of linear combinations and nonlinear functions of one variable, expression that use as few such linear combinations and functions of one variable as possible.

**We need to have at least one nonlinear function.** We want an expression for a probability density function. Probability density is always non-negative. Thus, we need expressions which are always non-negative. If we do not use nonlinear functions at all, then all we get is a linear expression, and each non-constant linear expression takes negative values. So, we need to have at least one nonlinear function.

**Fastest case: when we use only one nonlinear function.** Let us first consider the case when we use only one nonlinear function $f(x)$. In line with the general scheme, the input to this function can be a linear combination of the values $x_i$ and $y$, resulting in

$$z = f(b \cdot y + b_0 + b_1 \cdot x_1 + \ldots + b_n \cdot x_n).$$

After that, we can also have a linear combination of this value $z$ and the original inputs, resulting in

$$t = C \cdot f(b \cdot y + b_0 + b_1 \cdot x_1 + \ldots + b_n \cdot x_n) +$$

$$c \cdot y + c_0 + c_1 \cdot x_1 + \ldots + c_n \cdot x_n.$$

The linear terms

$$c \cdot y + c_1 \cdot x_1 + \ldots + c_n \cdot x_n$$

in this expression can cause the whole result to be negative, so we cannot have them, and we must have

$$t = C \cdot f(b \cdot y + b_0 + b_1 \cdot x_1 + \ldots + b_n \cdot x_n) + c_0.$$

To simplify this expression, we can introduce a new function

$$F(x) = C \cdot f(b \cdot x) + c_0.$$

In terms of this new function, the expression for $t$ takes the linear regression form

$$t = F(y - (a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n)),$$

for $a_i \stackrel{\text{def}}{=} -b_i/b$. Thus, in this case, the only option is linear regression.

**What if we can use two nonlinear functions.** In this case, one of possible expressions is the expression for the generalized logit regression: it uses two nonlinear functions $F(x)$ and $g(x)$ and two linear combinations.

For two nonlinear functions, in contrast to the case of a single nonlinear function, this is not the only option. In general, we can have a nonlinear function applied to some linear combinations, and then we apply linear combination to the result of this nonlinear function and to the original values:

$$C_1 \cdot f_1(b_1 \cdot y + b_{10} + b_{11} \cdot x_1 + \ldots + b_{1n} \cdot x_n) +$$

$$C_2 \cdot f_2(b_2 \cdot y + b_{20} + b_{21} \cdot x_1 + \ldots + b_{2n} \cdot x_n +$$

$$B_2 \cdot f_1(b_1 \cdot y + b_{10} + b_{11} \cdot x_1 + \ldots + b_{1n} \cdot x_n)) +$$

$$c_0 + c_1 \cdot x_1 + \ldots + c_n \cdot x_n.$$

Similarly to the case of a single nonlinear function, we can conclude that the linear terms in the right-hand side can be dismissed, and that the nonlinear terms can be simplified, resulting in:

$$F_1(y - (a_{10} + a_1 \cdot x_1 + \ldots + a_n \cdot x_n)) +$$

$$F_2(y - (a_{10} + a_1 \cdot x_1 + \ldots + a_n \cdot x_n) +$$

$$A_2 \cdot F_1(y - (a_{10} + a_1 \cdot x_1 + \ldots + a_n \cdot x_n))).$$

**What if we use three (or more) nonlinear functions.** One the cases when we use three nonlinear functions is exactly the case of the surrogate approach. If we are not satisfied with the accuracy of this approximation, we can, similarly to the 2-function cases, have other – more general – case of using three nonlinear functions. And if even these more general expressions do not lead to sufficient accuracy, we can use four (or more) nonlinear functions.

It is known that, in general, every continuous functions on a bounded domain can be approximated, with any given accuracy, by a neural network [1, 3, 4] – i.e., as we have mentioned, by compositions of linear functions and nonlinear functions of one variable. Thus, as we increase the number of nonlinear functions of one variable in the expression for the probability density, we can get approximations which are as accurate as we require.

## 4 Conclusions

We started this paper by formulating two questions: why surrogate approach works well, and what to do if we need a more accurate description of the corresponding probabilities. By using techniques form the foundations of neural networks, we get answers to both questions:

- The surrogate approach works well because it is, in some reasonable sense, the fastest. Namely, the fastest to compute are compositions of linear functions and nonlinear functions of one variable, and the largest part of computation time is spent on computing functions of one variable. If we are not satisfied with the accuracy of generalized logit regression – that uses two nonlinear functions – a natural next arrangement is to use three nonlinear functions – which is exactly the surrogate approach.

- If we are not satisfied with the accuracy of the surrogate approach – and of other arrangements that use three nonlinear functions – a natural next arrangement is to use four (or more) nonlinear functions. The universal approximation theory for neural networks implies that whatever accuracy we desire, we can achieve this accuracy if we use sufficiently many nonlinear functions.

## 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), EAR-2225395, and by the AT&T Fellowship in Information Technology.

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, and by a grant from the Hungarian National Research, Development and Innovation Office (NRDI).

The authors are thankful to Dungard Liu and Michael Pokojovy for valuable discussions.

## References

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

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

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

[4] V. Kreinovich. Arbitrary nonlinearity is sufficient to represent all functions by neural networks: a theorem, Neural Networks, 1991, Vol. 4, 381–383.

[5] V. Kreinovich, From traditional neural networks to deep learning: towards mathematical foundations of empirical successes, In: S. N. Shahbazova, J. Kacprzyk, V. E. Balas, and V. Kreinovich (eds.), Recent Developments and the New Direction in Soft-Computing Foundations and Applications: Selected Papers from the World Conference on Soft Computing, Baku, Azerbaijan, May 29–31, 2018, Springer, Cham, Switzerland, 2021, pp. 387–397.

[6] D. Liu, S. Li, Y. Yu, and I. Moustaki, Assessing partial association between ordinal variables: quantification, visualization, and hypothesis testing, *Journal of the American Statistical Association*, 2021, Vol. 116, No. 534, pp. 955–968.

[7] D. Liu and H. Zhang, Residuals and diagnostics for ordinal regression models: a surrogate approach, Journal of the American Statistical Association, 2018, Vol. 113, No. 522, pp. 845–854.

[8] K. S. Thorne and R. D. Blandford, Modern Classical Physics: Optics, Fluids, Plasmas, Elasticity, Relativity, and Statistical Physics, Princeton University Press, Princeton, New Jersey, 2021.