4-1-2022

When Is Deep Learning Better and When Is Shallow Learning Better: Qualitative Analysis

Salvador Robles Herrera
The University of Texas at El Paso, sroblesher1@miners.utep.edu

Martine Ceberio
The University of Texas at El Paso, mceberio@utep.edu

Vladik Kreinovich
The University of Texas at El Paso, vladik@utep.edu

Comments:
Technical Report: UTEP-CS-22-51
To appear in *International Journal of Parallel, Emergent and Distributed Systems*

RESEARCH ARTICLE

When Is Deep Learning Better and When Is Shallow Learning Better: Qualitative Analysis

Salvador Robles Herrera[a], Martine Ceberio[b], and Vladik Kreinovich[c]

Department of Computer Science, University of Texas at El Paso, El Paso, Texas 79968, USA

ARTICLE HISTORY
Compiled April 21, 2022

ABSTRACT
In many practical situations, deep neural networks work better than the traditional "shallow" ones, however, in some cases, the shallow neural networks lead to better results. At present, deciding which type of neural networks will work better is mostly done by trial and error. It is therefore desirable to come up with some criterion of when deep learning is better and when shallow is better. In this paper, we argue that this depends on whether the corresponding situation has natural symmetries: if it does, we expect deep learning to work better, otherwise we expect shallow learning to be more effective. Our general qualitative arguments are strengthened by the fact that in the simplest case, the connection between symmetries and effectiveness of deep learning can be theoretically proven.

KEYWORDS
Neural Networks; Deep Learning; Shallow Learning; Symmetry; Invariance.

1. Formulation of the Problem

Neural networks – general idea: a brief reminder. To design a flying machine, a natural idea is to look at creatures that fly – and to emulate them, e.g., by placing wings at the flying machine. Similarly, to have a machine that performs intellectual operations, a natural idea is to look at how these operations are performed in our brains.

In a brain, all data processing is performed by special cells called neurons that process electric signals. Each signal is a sequence of pulses; different information is conveyed by different frequencies, i.e., different numbers of pulses per second. A neuron has inputs several signals $x_1, \ldots, x_n$, and generates an output signal y. In the first approximation, the output signal has the form

$$y = s(w_1 \cdot x_1 + \ldots + w_n \cdot x_n - w_0), \tag{1}$$

where w_i are real numbers and $s(z)$ is a non-linear function called an *activation function*.

First, signals coming from our senses are processed by several neurons. Then, the output signals from these neurons are processed by other neurons, etc. As information is processed, the weights w_i of the neurons are adjusted – this is how we learn.

CONTACT V. Kreinovich. Email: vladik@utep.edu

This first approximation model is what is usually simulated in computers. The resulting computer model is known as a *artificial neural network*, or simply a *neural network*, for short.

Traditional (shallow) neural networks. One of the reasons why researchers started looking at the brain is that for many tasks like learning to recognize faces, humans are still better – and faster – than the most advanced computer programs. This happens in spite of the fact that the fastest biological neurons take several milliseconds to produce the output, while a computer can perform billions of operations per second. The main advantage of biological neural networks is that they are working in parallel: e.g., while each neuron participating in processing an image requires a few millisecond, during this time, millions of neurons perform some data processing, while even in most advanced high performance computers, we have at most thousands of computers working in parallel.

Since the main advantage of neural networks is speed, a natural idea is to form artificial neural networks which are as fast as possible. In general, as we have mentioned, the signals go through several consecutive stages:

- at first stage, several neurons simultaneously process,
- then, on the second stage, the outputs of these neurons are simultaneously processed by yet other neurons, etc.

The more such stages we have, the longer data processing takes. Thus, to speed up computations, it is necessary to use the smallest possible number of stages.

With a single stage, we only get functions of type (1), and not all functions of several variables can be represented in this form. So, we need at least two processing stages. It turned out (see, e.g., [1,2]) that with two layers, we can already approximate any function with any given accuracy – moreover, we can do it even if on the second layer, we use the fastest-to-compute linear transformation of the inputs, i.e., if we return a linear combination

$$y = W_1 \cdot y_1 + \ldots + W_K \cdot y_K - W_0, \tag{2}$$

where the values $y_1, \ldots, y_K$ are the results of the first stage of data processing:

$$y_k = s(w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n - w_{k0}). \tag{3}$$

One of the proofs of this results comes from the fact that – as Newton showed with his prism experiment – every light can be represented as a linear combination of pure colors – i.e., in mathematical terms, sinusoids. A similar representation as a linear combination of sinusoids – known as Fourier transform – is possible for any continuous functions, and this representation has exactly the form (2)-(3), with $s(z) = \sin(z)$.

The resulting 2-stage neural networks, with linear second stage, are what is now called *shallow* neural networks.

Learning: a general idea. What do biological neural networks do? For example, they allow us, by looking at a picture, to see if this is a picture of a cat or of a dog. In this case, the inputs are intensities x_i of different pixels of the picture, and the output y is 0 or 1 depending on whether it is a cat or a dog. In more complicated cases, when the brain solves a mathematical problem, x_i are inputs to this problem (e.g., two numbers 3 and 4 that we want to add), and y is the desired result (in this case, $3 + 4 = 7$).

The brain of a newborn baby cannot yet solve any of these problems, it needs to be trained. Training means that we show a baby different pictures of pets and indicate which of them are cats and which are dogs. Eventually, the baby learns this.

In precise terms, learning means that we are looking for the parameters w_{ki} and W_k of the neural network for which, on the known examples, the match between the outputs of the neural network and the desired values is the best. Specifically, we have several examples $p = 1, \ldots, P$ in which we know both the inputs $x_1^{(p)}, \ldots, x_n^{(p)}$ and the desired output $y^{(p)}$. We want to find the values w_{ki} and W_k for which the outputs

$$y_{NN}^{(p)} = W_1 \cdot y_1^{(p)} + \ldots + W_K \cdot y_K^{(p)} - W_0$$

of the neural network (NN) are close to the desired values $y^{(p)}$, where

$$y_k^{(p)} = s\left(w_{k1} \cdot x_1^{(p)} + \ldots + w_{kn} \cdot x_n^{(p)} - w_{k0}\right).$$

There are efficient algorithm for determining the values w_{ki} and W_k, these algorithms are used in artificial neural networks.

Comment. To be more precise, we split the known examples into training examples and testing examples, find the weights based only on the training examples, and then gauge the quality of the result by testing it on the testing examples.

Enter deep learning. In the last decades, it was shown that in many practical applications, we get a better approximation if we use neural networks with more than 2 stages; see, e.g., [2]. Such neural networks are known as *deep*.

Sometimes, deep learning is better, sometimes, shallow learning is better. The more stages we have, the more neurons we will need, and thus, the values of more parameters we will need to determine based on the data. In many practical situations, we have only a relatively small number of data points. A good example of such a situation is the analysis of strong volcanic eruptions: luckily, there are not so many of them. In such situations, we do not have enough data to train a deep network, so if we want to use neural networks, shallow ones are our only option.

When we have more data, in principle, we can apply both deep learning and shallow learning. In many cases, deep learning works better, but in some cases, shallow learning leads to better results.

Resulting question. At present, there seems to be no good understanding of when deep learning works better and when shallow learning works better. As a result, now, the decision on which type of neural network to use is made by time-consuming trial and error. It is therefore desirable to come up with an understanding of when each of these techniques is better.

What we do in this paper. In this paper, we make a first step towards answering this question: namely, we provide a qualitative understanding. On a simplified example, we show that this qualitative understanding makes sense.

2. Analysis of the Problem: Deep Learning vs. Shallow Learning

Main difference between shallow and deep learning. In both shallow learning and deep learning, we start with basic computational units (neurons) for which the

function describing how the unit's output depends on its inputs takes the form (1). To get more complex dependencies, we combine these basic functions. From this viewpoint, the main difference between shallow and deep learning is in how we combine them:

- In shallow learning, we have only two data processing layers, with the second one linear. Thus, we simply take a linear combination (2) of several functions of type (1).
- In contrast, in deep learning, we have several nonlinear data processing layers. So, the outputs of nonlinear neurons become inputs to other neurons, i.e., from mathematical viewpoint, we also consider *compositions* of nonlinear functions of type (1).

Indeed, if we first transform the input x into the output $y = f_1(x)$ and then use another unit to transform the resulting signal y into a new signal $z = f_2(y)$, then this procedure transforms the original input x into the value $z = f_2(f_1(x))$. In other words, the function corresponding to two consequent processing stages is exactly the composition of the functions corresponding to each of these stages.

So when is shallow learning better and when is deep learning better: brainstorming. As we have mentioned earlier, for some classes of practical problems – i.e., in mathematical terms, for some classes of functions that we want to approximate by neural networks – deep learning works better, while for other classes, shallow learning works better.

Based on the above description of the difference between shallow and deep learning, it is reasonable to make the following informal conclusion

- Shallow learning is expected to work better if the corresponding class of functions F is closed under linear combinations, i.e., if this class F has the property that:
  - if F contains functions $f_1(x), \ldots, f_n(x)$,
  - then it should contain all their linear combinations

$$W_1 \cdot f_1(x) + \ldots + W_n \cdot f_n(x).$$

- In contrast, deep learning is expected to work better if the corresponding class of functions F is closed under composition, i.e.: if this class F has the property that:
  - if F contains functions $f_1(x)$ and $f_2(x)$,
  - then it should contain their composition $f_2(f_1(x))$.

Let us describe this difference in mathematical terms. In the shallow learning case, we have classes which are closed under linear combination. These classes are well known in mathematics, they are called *linear spaces*.

In the deep learning case, we have classes which are closed under composition. Such classes are known as *transformation semigroups*. An important particular case is *transformation groups* – they describe the case when all transformations are reversible, and the inverse transformation also belongs to the same class F.

3. So When Is Deep Learning Better: A General Answer

Our conclusion. Based on the above argument, we can make the following conclusion:

- if the set of approximated functions is closed under linear combinations – i.e., forms a linear space – then, on functions from this class, we expect shallow learning to work better, and
- if the set of approximated functions is closed under composition – i.e., forms a transformation semigroup – then, on functions from this class, we expect deep learning to work better than shallow learning.

Comment. Of course, this is an approximate qualitative conclusion, since deep networks also include linear combinations.

Transformation semigroups are ubiquitous. At first glance, the property to be closed under composition – i.e., to form a transformation semigroup – sounds very specific. However, in practice, such sets are ubiquitous, and this ubiquity is easy to explain.

In physics, it is well understood (see, e.g., [3,4]) that transformations play a fundamental role in how we gain knowledge. In effect, all our knowledge is based on appropriate transformations.

Indeed, how do we know that if we drop a pen, it will fall down with acceleration of 9.81 m/sec^2? Someone observed it in one location. Then this person moved to a different location – i.e., performed a shift $x \mapsto x + x_0$ of his/her coordinates, repeated the same experiment and got the same result. Then, this person turned around by some angle, repeated the experiment, and got the same result. This person also repeated this same experiment at some future moment of time, i.e., at a moment of time obtained by a shift in time $t \mapsto t + t_0$, and got the same result,

So, this person naturally concluded that the result of this dropping-a-pen experiment does not change if we apply a shift in space, a shift in time, and/or rotation. In mathematical terms, we say that this phenomenon is *invariant* or *symmetric* with respect to these transformations, and the transformations themselves are called *symmetries*. Since our current situation can be obtained from the original one by some shifts and rotations, we can therefore predict that if we drop a pen right now, in our current on-Earth location, it will exhibit the exact same behavior: namely, it will start falling down with the acceleration 9.81 m/sec^2.

In general, how do we know, e.g., that coughing and sneezing are usually symptoms of flu, cold, or allergy? Because we observed similar situations in the past, and this is what they turned out to be. And what does "similar" mean? The time is different, the person may be different, the location may be different – but similar means that we can perform some transformations like shifts, rotations, renaming people, transformations with respect to which medical conditions are invariant, and we can get the new situations – exactly or approximately – from the old ones.

How do we know that a resistor in our lab will follow Ohm's law? Because this law was confirmed at different locations at different moments of time, and we thus learned that this law does not change if we move to a different location at a different moment of time.

The transformations do not have to be as simple as shifts and rotations. For example, for many properties, if we replace each particle with its anti-particle – electron with positron, proton with anti-proton, etc. – we will get the same physical phenomena: e.g., positrons will combine with anti-protons to form anti-Hydrogen atoms whose physical properties will be largely the same as of the usual Hydrogen.

Similarly, if we place a small-size model of an airplane in a wind tunnel, it will behave the same way as the real-size airplane – this is how airplane designs were tested before it became possible to run accurate computer simulations.

In physics, transformations that preserve some properties of the object of study are known as *symmetries*. This name comes from the fact that in geometry, e.g., if we have a set invariant with respect to rotations around an axis – e.g., a round cylinder – we say that this set is symmetric with respect to this rotation. In physics, the word "symmetry" is used for general transformations, not necessary geometric one: e.g., replacing each particle with the corresponding anti-particle is also called a symmetry.

Our answer reformulated. In view of this, our answer can be reformulated as follows:

- if there is a symmetry, then deep learning should work better, and
- if there is no symmetry, then shallow learning should work better.

This explain why deep learning is so successful. Since, as we have mentioned, symmetries are ubiquitous, this explain why areas in which deep learning work better are also ubiquitous.

4. Let Us Check Our Answer on a Simple Case

Simple case: a description. To check whether our answer to the shallow vs. deep question makes sense, let us check it on a simple example.

The simplest activation function. To come up with such an example, we need, among other things, to select a simple non-linear activation function $s(x)$. Since we are talking about computations, by the simplest, we mean the simplest to compute. In a computer, in effect, the only hardware supported operations are addition and multiplication. Any other computation is performed as a sequence of additions and multiplications. In other words, whatever function we want to compute, what we actually compute is a composition of additions and multiplications – i.e., a polynomial. For example, when we ask a computer to compute $\exp(x)$, what the computer will actually compute is the sum of the first few terms of the Taylor series for this function, i.e., the expression

$$1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!}.$$

The simplest polynomials are linear, the next simplest are quadratic. So, the simplest nonlinear activation function is a quadratic function $s(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2$.

The simplest deep-learning-type architecture. The main difference between shallow and deep learning is that in deep learning, outputs from nonlinear neurons become inputs to other nonlinear neurons. This means that, in deep learning, we need at least two nonlinear neurons.

From this viewpoint, the simplest deep neural network is when we have two nonlinear neurons, one processes the input(s), and the other one processes the output of the first neuron.

The simplest number of inputs. The more inputs, the more complex the situation. So, the simplest case is when we have a single input.

How the resulting simplest deep neural network looks like. Let us summarize the above selections. We have a neural network with one input and two neurons each

of which is described by a quadratic activation function. The output of the first neuron serves as the input to the second neuron.

Let us describe this in precise terms. The first neuron transforms the input x into the value $y = a_0 + a_1 \cdot x + a_2 \cdot x^2$, for some coefficients a_i. The second neuron then transforms the resulting signal y into the signal $z = b_0 + b_1 \cdot y + b_2 \cdot y^2$, for some coefficients b_i.

What function is computed by this simplest network. Substituting the expression for y into the formula for z, we conclude that

$$z = b_2 \cdot \left(a_0 + a_1 \cdot x + a_2 \cdot x^2\right)^2 + b_1 \cdot \left(a_0 + a_1 \cdot x + a_2 \cdot x^2\right) + b_0. \tag{4}$$

This function is a composition of two quadratic polynomial and is, thus, a polynomial of 4-th order.

What are the corresponding symmetries. In 1-D space, reasonable geometric transformations are shifts and reflections with respect to a point.

A function is invariant with respect to a shift $x \mapsto x+T$ means that $f(x) = f(x+T)$, i.e., that the function is periodic with period T. One can easily check that non-constant polynomials cannot be periodic. So, since we are considering polynomials, the only transformation with respect to which a polynomial can be invariant are reflection with respect to a point a. Reflection means that for each original point x, we take a new point x' which is located at the same distance from a as the original point x, but at the opposite side of the point a. In other words, we have $x' - a = -(x - a)$, i.e., equivalently, $x' = 2a - x$.

Main result of this section. For the above example, we get the following result.

Proposition. *A 4-th order polynomial can be represented as a composition of two quadratic polynomials if and only if this polynomial is invariant with respect to some reflection.*

Comment. In other words, a 4-th order polynomial can be computed by the above simplest deep neural network if and only if this polynomial is symmetric. This is perfectly in line with what we claimed in the previous section – that deep learning should work better if there is a symmetry.

Proof of the Proposition.

1°. Let us first prove that if a 4-th order polynomial $z(x)$ is symmetric with respect to reflection against some point a, then it can be represented as a composition of two quadratic functions.

Indeed, let us introduce a new variable $t \stackrel{\text{def}}{=} x - a$, for which $x = t + a$. In terms of the new variable, the function $z(x)$ takes the form $Z(t) \stackrel{\text{def}}{=} z(t+a)$. This function $Z(t)$ is a 4-th order polynomial, so $Z(t) = A_0 + A_1 \cdot t + A_2 \cdot t^2 + A_3 \cdot t^3 + A_4 \cdot t^4$ for some coefficients A_i. In terms of t, symmetry $z(2a - x) = z(x)$ takes the form $Z(t) = Z(-t)$ for all t, i.e.,

$$A_0 + A_1 \cdot t + A_2 \cdot t^2 + A_3 \cdot t^3 + A_4 = A_0 - A_1 \cdot t + A_2 \cdot t^2 - A_3 \cdot t^3 + A_4.$$

If two polynomials are equal, this means that all their coefficients must coincide, so we must have $A_1 = A_3 = 0$, hence $Z(t) = A_0 + A_2 \cdot t^2 + A_4 \cdot t^4$ and thus,

$$z(x) = Z(x - a) = A_0 + A_2 \cdot (x - a)^2 + A_4 \cdot (x - a)^4,$$

i.e., $z = A_0 + A_2 \cdot y + A_4 \cdot y^2$, where

$$y \stackrel{\text{def}}{=} (x-a)^2 = x^2 - 2a \cdot x + a^2.$$

So, the function $z(x)$ can indeed be represented as a composition $z(x) = z(y(x))$ of two quadratic functions $z(y)$ and $y(x)$.

2°. Let us now prove that if a 4-th order polynomial can be represented as a composition of two quadratic function, then it is symmetric with respect to some reflection. Indeed, let's start with

$$y = a_0 + a_1 \cdot x + a_2 \cdot x^2$$

, where $a_2 \neq 0$. The standard transformation used in deriving the formula for the solutions of a quadratic equation leads to:

$$a_2 \left(x^2 + \frac{a_1}{a_2} \cdot x + \frac{a_0}{a_2} \right) = a_2 \left(\left(x + \frac{a_1}{2 \cdot a_2} \right)^2 + \frac{a_0}{a_2} - \left(\frac{a_1}{2 \cdot a_2} \right)^2 \right).$$

So if we take $t \stackrel{\text{def}}{=} x - a$, where

$$a \stackrel{\text{def}}{=} -\frac{a_1}{2 \cdot a_2},$$

we get

$$y = a_2 \left(t^2 + \frac{a_0}{a_2} - \left(\frac{a_1}{2 \cdot a_2} \right)^2 \right),$$

where $\frac{a_0}{a_2} - \left(\frac{a_1}{2 \cdot a_2} \right)^2$ is a constant.

Clearly if we replace t with $-t$ the result y remains the same. Thus, $z(y)$ will also remain the same.

In terms of $x = t + a$, the transformation $t \mapsto -t$ takes the form $x \mapsto 2a - x$, i.e., is reflection against the point a. Thus, the dependence $z(x)$ does not change – i.e., is symmetric – if we apply this reflection.

The proposition is proven.

Acknowledgement(s)

The authors are greatly thankful to Dr. Adamatzky for his encouragement.

Funding

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), and HRD-1834620 and HRD-2034030 (CAHSI Includes), 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).

References

[1] Bishop CM. Pattern recognition and machine learning. New York: Springer; 2006.

[2] Goodfellow I, Bengio Y, Courville A. Deep learning. Cambridge, Massachusetts: MIT Press; 2016.

[3] Feynman R, Leighton R, Sands M. The Feynman lectures on physics. Boston, Massachusetts: Addison Wesley; 2005.

[4] Thorne KS, Blandford RD. Modern classical physics: optics, fluids, plasmas, elasticity, relativity, and statistical physics. Princeton, New Jersey: Princeton University Press; 2017.