# Computational Paradox of Deep Learning: A Qualitative Explanation

Jonatan Contreras[1], Martine Ceberio[1], Olga Kosheleva[1], Vladik Kreinovich[1], and Nguyen Hoang Phuong[2]
[1]University of Texas at El Paso, El Paso, Texas 79968
jmcontreras2@miners.utep.edu, mceberio@utep.edu, olgak@utep.edu, vladik@utep.edu
[2]Division Informatics, Math-Informatics Faculty, Thang Long University
Nghiem Xuan Yem Road, Hoang Mai District
Hanoi, Vietnam, nhphuong2008@gmail.com

## 1. Regression/machine learning: a brief reminder

- In many practical situations, we need to find a dependence $y = f(x_1, \ldots, x_n)$ between different quantities.
- Available information: on several cases $k = 1, 2, \ldots, K$ we know the values $x_1^{(k)}, \ldots, x_n^{(k)}$ and $y^{(k)}$ of all related quantities.
- In statistics, this problem is known as *regression*.
- In computer science, it is known as *machine learning*.
- The usual approach to solving this problem is:
  - to select a family of functions $f(x_1, \ldots, x_n, c_1, \ldots, c_m)$ and
  - to select the values of the parameters $c_1, \ldots, c_m$ for which, for all $k$ from 1 to $K$, we have

$$y^{(k)} \approx f\left(x_1^{(k)}, \ldots, x_n^{(k)}, c_1, \ldots, c_m\right).$$

- Sometimes, we select an explicit formulation of this family.

## 2. Regression/machine learning: a brief reminder (cont-d)

- E.g., in linear regression, we select linear functions:

$$f(x_1, \ldots, x_n, c_1, \ldots, c_{n+1}) = c_1 \cdot x_1 + \ldots + c_n \cdot x_n + c_{n+1}.$$

- In other cases, we select the family of all quadratic functions, etc.
- In many machine learning techniques, the corresponding family is described indirectly.
- E.g., in neural networks, the values $c_i$ are weights of connections between different neurons.
- It is known that the efficiency of regression/machine learning depends on the proper section of the corresponding family.
- In particular, it depends on the number of parameters $m$.

## 3. In general, computational complexity increases with the number of unknowns

- In general, the larger the number of unknowns, the more computational effort we need to solve the corresponding problem.
- For example, optimizing a function of one variable is easier than optimizing a function of two or three variables.
- Solving a single equation is usually computationally easier than solving a system of several equations, etc.

## 4. Under-fitting and over-fitting: traditional statistical approach

- The efficiency of regression/machine learning depends on the number of parameters $m$.
- If we select the value $m$ to be too small, we may not get a good fit.
- E.g., if we only consider linear regression and the actual dependence is highly non-linear.
- This is known as *under-fitting*.
- On the other hand:
  - if we select the value $m$ to be too high, e.g., if we choose $m \geq K$,
  - then we get a system of $K$ equations with $m \geq K$ unknowns.
- In general, in such cases, we can get exact equality in all $K$ equations:

$$y^{(k)} = f\left(x_1^{(k)}, \ldots, x_n^{(k)}, c_1, \ldots, c_m\right).$$

- For $m > K$, we have more unknowns than equations.

## 5. Under-fitting and over-fitting (cont-d)

- So we have several possible solutions for which there is exact equality.
- The problem is that the measurement results $x_1^{(k)}, \ldots, x_n^{(k)}$ and $y^{(k)}$ come with random measurement errors.
- The exact fit means that we follow this randomness.
- Thus, we get a not very good prediction accuracy.
- For example, suppose that the actual value of $y$ is constant $y = 1$.
- We try to exactly fit the measurement results $y^{(1)} = 1.0$ and $y^{(2)} = 1.1$ corresponding to $x_1^{(1)} = 0$ and $x_1^{(2)} = 1$.
- We assume that the dependence is linear: $y = c_1 \cdot x_1 + c_2$.
- Then we get the equations

$$y^{(1)} = c_1 \cdot x^{(1)} + c_2 \text{ and } y^{(2)} = c_1 \cdot x^{(2)} + c_2.$$

## 6. Under-fitting and over-fitting (cont-d)

- These equations lead to $c_1 = 0.1$ and $c_2 = 1$, i.e., to the formula $y = 0.1 \cdot x + 1$.
- For large $x$, the predicted value can be very different from the desired value $y = 1$.
- This phenomenon is known as *over-fitting*.

## 7. First (accuracy-related) paradox of deep learning: going beyond supposed over-fitting increases prediction accuracy

- At present, the most efficient machine learning techniques are the techniques of deep learning.
- These techniques require a large amount of data to be successful.
- In situations when we do not have that many available data points, other machine learning techniques work better.
- Interestingly, state-of-the-art machine learning techniques often achieve their successes when:
  - the number of parameters $m$ is much larger than
  - the number $K$ of data points.
- This is contrary to what the traditional statistics predicts.
- Researchers have studied this phenomenon in detail by gradually increasing $m$.

## 8. First (accuracy-related) paradox of deep learning (cont-d)

- They found that as $m$ increases beyond $K$:
  - at first, the prediction accuracy decreases – exactly as predicted by traditional statistics,
  - however, as the value $m$ increases further, the prediction accuracy starts increasing again.
- A mathematical explanation of this paradox has been proposed by Belkin et al.

## 9. Remaining problem: computational paradox of deep learning

- There is another deep-learning-related paradox for which there is, at present, no satisfactory solution.
- Specifically, it turned out that as the number $m$ of parameters increases:
  - solving the corresponding system of approximate equations,
  - which usually means minimizing some functional (like least squares) $F(c)$, where $c \stackrel{\text{def}}{=} (c_1, \ldots, c_m)$
  - that describes the discrepancy between the left- and right-hand sides,
  - becomes computationally easier.
- In this paper, we provide a possible qualitative explanation for this computational paradox.
- This explanation is based on the following two known results.

## 10. First known result: uniqueness makes computations easier

- It is known that, in general, uniqueness makes solutions easier.
- For many general computational problems, there is an algorithm that solves all the cases in the which the solution is unique.
- This is true for all kinds of problems – optimization problems, solving systems of equations, etc.
- On the other hand, it can be proven that:
  - no algorithm is possible
  - that would solve all the cases in which the problem has exactly two solutions.
- There exist good arguments that a similar phenomenon occurs for problems for which a general algorithm is possible.
- These arguments show that, in general, cases for which there exists a unique solution are easier to solve.

## 11. Second known result: a random function attains its optimum at a single point

- The second result is related to the study of random functions, i.e., reasonable probability measures on reasonable sets of functions $F(c)$.
- An important feature of such probability measures is that:
  - for each reasonable numerical characteristic $v(F)$ of a function $F(c)$
  - such as its integral or its maximum or minimum over a certain area,
  - this characteristic should also be "random" in the intuitive sense of this word.

## 12. Second known result (cont-d)

- In particular, this means:
  - that for each real number $r$, the probability that we have $v(F) = r$ is equal to 0, and
  - that for every two different characteristics $v \neq v'$, the probability that they have the same value, i.e., that $v(F) = v'(F)$, is also equal to 0.
- Let us show how this applies to minima of random functions.
- Suppose that a function $F(c)$ attains its minimum value $m$ for each two different inputs $s \neq t$: $F(s) = F(t) = \min\limits_{c} F(c)$.
- Then, we can find a hyperplane $H$ with rational coefficients that separates these inputs $s$ and $t$:
  - the input $s$ is in one of the half-spaces $H_-$ corresponding to this hyperplane, while
  - the input $t$ is in another of these two half-spaces $H_+$.

## 13. Second known result (cont-d)

- Since $s \in H_-$ and $F(s) = m$, we have
$$m = \min_c F(c) \leq \min_{c \in H_-} F(c) \leq F(s) = m.$$
- Thus, $\min_{c \in H_-} F(x) = m$.
- Similarly, due to $t \in H_+$ and $F(t) = m$, we have $\min_{c \in H_+} F(c) = m$.
- Thus, we have $\min_{c \in H_-} F(c) = \min_{c \in H_+} F(c)$.
- The probability that two characteristics $\max_{c \in H_-} F(c)$ and $\max_{c \in H_+} F(c)$ have equal values is 0.
- There are countably many rational numbers, so there are countably many hyperplane with rational coefficients.
- For each of them, the probability of above equality is zero.
- Thus, the probability that this equality happens for some hyperplane with rational coefficients is also 0.

## 14. Second known result (cont-d)

- Indeed, the union of countably many sets of probability measure 0 also has probability measure 0.
- Thus, the probability that a function attains its minimum at two (or more) different inputs is 0.
- What about a *random* function, i.e., a function:
  - that does not belong to any definable set of measure 0,
  - or, alternative, that satisfies every law that is true for almost all functions.
- Such function cannot attain its minimum at two or more different points.
- So, a random function attains its minimum at a single point.
- Now, we are ready to present our explanation.

## 15. Resulting explanation

- In the usual statistical case:
  - when the number of parameters exceeds the number of unknowns,
  - we have a non-unique solution to the corresponding fitting problem.
- In deep learning, we do not simply make a fit, there is a lot of randomness involvef.
- E.g., initial value of all the coefficients are random.
- This randomness is not just a feature of a specific training algorithm, it is inevitable.

## 16. Resulting explanation (cont-d)

- Indeed, otherwise:
  - if we start with the same deterministic values of the weights, and
  - we use some deterministic algorithm for training,
  - then all neurons in each layer will have the same weights,
  - so their outputs will simply uselessly duplicate each other.
- Because of this randomness, the actual objective function corresponding to a neural network is random.
- Thus, it attains its minimum at only one point.
- So:
  - in the traditional regression case, we have a problem with multiple solutions, while
  - in the neural network case, we have a problem with a single solution.

## 17. Resulting explanation (cont-d)

- We have mentioned that problems with a single solution are easier to solve than problems with multiple solutions.
- So, this explains, on the qualitative level, why:
  - optimization related to training a neural network turns out to be computationally easier than
  - the optimization related to the usual multi-parameter regression.

## 18. Specific Explanation Related to Gradient-Based Training of Neural Networks

- For neural networks, the above general explanation can be supplemented by specific details.
- These details are based on the fact that training of a neural network is based on gradient descent.
- On each iteration, we replace the previous values $c_i$ of all the parameters by a new value $c_i' = c_i + \Delta c_i$, where $\Delta c_i = -\alpha \cdot \dfrac{\partial F}{\partial c_i}$.
- Here, $\alpha$ is some constant.
- After this change, the value of the objective function changes from the previous value $F(c_1, \ldots, c_m)$ to the new value

$$F(c_1', \ldots, c_m') = F(c_1, \ldots, c_m) + \Delta F, \text{ where}$$

$$\Delta F \stackrel{\text{def}}{=} F(c_1', \ldots, c_m') - F(c_1, \ldots, c_m) =$$

$$F(c_1 + \Delta c_1, \ldots, c_m + \Delta c_m) - F(c_1, \ldots, c_m).$$

## 19. Specific Explanation Related to Gradient-Based Training of Neural Networks (cont-d)

- The changes $\Delta c_i$ are usually small.
- So, to estimate $\Delta F$, we can:
  - expand the right-hand side of the above formula in Taylor series in terms of $\Delta c_i$ and
  - keep only linear terms in this expansion.
- As a result, we get $\Delta F = \sum\limits_{i=1}^{m} \dfrac{\partial F}{\partial c_i} \cdot \Delta c_i$.
- Substituting the expression for $\Delta c_i$ into this formula, we conclude that $\Delta F = -\alpha \cdot \sum\limits_{i=1}^{m} \left(\dfrac{\partial F}{\partial c_i}\right)^2$.

## 20. Resulting explanation

- As we have mentioned:
  - to fit $K$ equalities,
  - it is sufficient to use $K$ parameters $c_1, \ldots, c_K$.
- In this case, after each iteration of the gradient descent, the value of the objective function decreases by $\Delta F_K = -\alpha \cdot \sum\limits_{i=1}^{K} \left( \dfrac{\partial F}{\partial c_i} \right)^2$.
- If we use all $m > K$ parameters, then the resulting decrease in the value of the minimized objective function $F(c)$ is even larger.
- The larger $m$, the larger this decrease.
- So, indeed:
  - if we increase the number of coefficients,
  - gradient descent becomes much more efficient than for smaller number of coefficients.
- This is exactly what was empirically observed.

## 21. 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), and
  - HRD-1834620 and HRD-2034030 (CAHSI Includes).
- It was also supported 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.