How to Make a Neural Network Learn from a Small Number of Examples – and Learn Fast: An Idea

Vladik Kreinovich[1] and Chitta Baral[2]
[1]Department of Computer Science University of Texas at El Paso
El Paso, Texas 79968, USA, vladik@utep.edu
[2]Department of Computer Science and Engineering, Arizona State University
Tempe, Arizona 85287-5406, USA, chitta@asu.edu

1. What is machine learning: a brief reminder

- In many practical situations, we know that a physical quantity y is largely determined by the values of the quantities $x_1, \ldots, x_n$.
- Here are some examples.
- We know that:
  - tomorrow's temperature at some location
  - is largely determined by today's values of temperature, humidity, wind speed, etc., at this and nearby locations.
- We know that:
  - the location of all the planets at some future moment of time
  - is uniquely determined by their current locations and velocities, etc.

2. What is machine learning: a brief reminder (cont-d)

- In some cases, we know the algorithm $y = f(x_1, \ldots, x_n)$ that predicts y based on the values $x_1, \ldots, x_n$.
- This happens, e.g., in celestial mechanics that describes the planets' motion.
- In many other situations, however, we do not know such an algorithm.
- For example:
  - based on a clear picture of an animal,
  - we can tell – in almost all cases – whether it is picture of a cat or not.
- In other words, we know that:
  - the binary variable $y =$"is a cat" (which is equal to 1 if it is a cat an 0 if not)
  - is uniquely determined by the intensities $x_1, \ldots, x_n$ of different pixels forming the image.

3. What is machine learning: a brief reminder (cont-d)

- However, we do not have an algorithm for such a detection.
- In general, we know that y is (largely) uniquely determine by the values x_i.
- In mathematical terms, this means that there is a function $y = f(x_1, \ldots, x_n)$ that determines y based on x_i.
- This is how the notion of a function is defined in mathematics, as a relation in which:
  - the value y is uniquely determined
  - by the values $x_1, \ldots, x_n$.
- However, we do not know this function.
- We do not know how to compute its values.
- What we *do* know in such situations is several (K) examples in which we know the values both of the values x_i and y.

4. What is machine learning: a brief reminder (cont-d)

- In other words, for each $k = 1, \ldots, K$ we know the *patterns* $(x_1^{(k)}, \ldots, x_n^{(k)}, y^{(k)})$ for which

$$y^{(k)} \approx f(x_1^{(k)}, \ldots, x_n^{(k)}) \text{ for the unknown function } f(x_1, \ldots, x_n).$$

- For example, we have K images of different animals, and for each of these images, we know:
  - whether it is an image of a cat ($y^{(k)} = 1$)
  - or it is an image of some other animal ($y^{(k)} = 0$).
- In general, based on such information, we want to reconstruct the desired function $f(x_1, \ldots, x_n)$.
- This will enable us in the future cases:
  - when we know the values $x_1, \ldots, x_n$,
  - to predict y as $f(x_1, \ldots, x_n)$.

5. What is machine learning: a brief reminder (cont-d)

- The problem of reconstructing a function based on the patterns is known as the problem of *machine learning*.
- There are many effective machine learning tools.
- At present, the most promising is *neural networks* (to be more precise, deep learning, i.e., a neural network with multiple layers).
- Why they are effective is (somewhat) clear
- On the theoretical level, the effectiveness of neural networks is (partly) justified by the existence of several *universal approximation* results.
- According to these results, crudely speaking:
  - any reasonable function can be approximated, with desired accuracy,
  - by an appropriate neural network.

6. What is machine learning: a brief reminder (cont-d)

- On the intuitive level, the effectiveness of neural networks can be explained by the fact that:
  - artificial neural networks are similar to biological neural networks,
  - and biological neural networks is how we humans learn from examples.

7. Main limitation of current machine learning techniques

- Deep learning algorithms have spectacular successes:
  - they enable computers to play chess and Go much better than humans,
  - they help us solve many practical problems.
- However, in comparison to humans, the current deep learning techniques have a severe limitation.
- To a human being, it is sufficient to have a few examples (patterns).
- Then, without practically any delay we can learn to distinguish cats from dogs, etc.
- In contrast, a deep neural network requires a large number of examples – thousands and even millions – to start producing reasonable results.
- Even on modern super-fast high-performance computers, it takes a long time to train a neural network.

8. Main limitation of current machine learning techniques (cont-d)

- To be fair, it should be mentioned that:
  - while a neural network usually takes a very long time to *train*,
  - once it is trained, it *produces its results very fast.*
- For example:
  - even in many applications involving solution to differential equations, where algorithms are known,
  - it is now much faster to use a trained neural network to produce the solution than to use the known algorithms.

9. What we do in this talk

- In view of the above limitation, it is desirable to come up with a machine learning tool that will enable us:
  - to learn from a small number of examples,
  - and to learn fast, just like we humans do.
- In this talk, we describe a proposal for designing such a tool.
- We also speculate that this may already be happening:
  - for in-context learning systems such as GPT3 and ChatGPT,
  - systems that produce very reasonable answers to queries already after 5-10 examples.

10. Let us describe what we want in precise terms

- What we would like to have is a *universal* computer system that:
  - given a small number of examples from any area and a new input,
  - would generate a reasonable answer to this new input.
- In mathematical terms, this means that:
  - this system should take, as an input, the tuple X that contain all the input information, i.e.,

$$X = (x_1^{(1)}, \ldots, x_n^{(1)}, y^{(1)}, \ldots, x_1^{(K)}, \ldots, x_n^{(K)}, y^{(K)}, x_1, \ldots, x_n); \quad (1)$$

  - and this system should return the value y corresponding to the input $x_1, \ldots, x_n$.

11. Let us describe what we want in precise terms (cont-d)

- For example:
  - we should show this system several images of a cat (i.e., images $x_1^{(k)},\ldots,x_n^{(k)}$ for which $y^{(k)}=1$),
  - several images of other animals (i.e., images $x_1^{(k)},\ldots,x_n^{(k)}$ for which $y^{(k)}=0$), and
  - a new image $x_1,\ldots,x_n$.
- The system should then decide whether this new image is the image of a cat ($y=1$) or of some other animal ($y=0$).
- Also:
  - we should show, to this same system, many images of vehicles indicating which of them are trucks and which are not trucks,
  - then show this system a new picture of a vehicle,
  - and this system should tell us whether this new image is truck or not a truck.

12. What do we know about this problem

- We know that we humans have this universal ability:
  - given the corresponding input X,
  - to produce the corresponding value y – whether it is about cars, about cars, or about something else.
- In other words, we know that the values forming the tuple X largely uniquely determine the desired value y.
- In mathematical terms, as we have mentioned, this means that there exists a function $y = F(X)$ that:
  - takes, as input, a tuple of type X and
  - returns the corresponding value y.
- This formulation leads to the following natural idea.

13. Natural idea

- We have an unknown function $F(X)$.
- We know that neural networks are universal approximators, i.e., that:
  - for each reasonable function and each desired accuracy,
  - there exists a neural network that approximates the given function with the desired accuracy.
- So why not use a neural network to approximate this unknown function $F(X)$?

14. Important comment

- In the above statement, we overlooked a somewhat minor but important point.
- Namely, the universal approximation theorem was proven for the case when the number of inputs is fixed.
- In our case, the number of inputs N forming a tuple X may be different depending on:
  - how many examples K we have and
  - how many inputs n are there in each example.
- In general:
  - we have K examples with $n+1$ values in each of then, and
  - we have n values of a new example.
- So, we have the total of $N = K \cdot (n+1) + n$ numbers.

15. Important comment (cont-d)

- So, to be able to apply the universal approximation result, let us limit ourselves to the cases when we have:
  - a fixed number of examples K and
  - the fixed number of inputs n in each example.
- For example, we can limit ourselves to cases when we have $K = 10$ examples with $n = 4$ inputs each.
- Then, in each such case, we have tuples X with $N = 10\cdot(4+1)+4 = 54$ inputs.

16. Resulting proposal

- Suppose that we want to design a computer system that will learn – in all possible application areas – after being presented K examples.
- Ideally, we should make this system really universal, i.e., it should be applicable to all possible input sizes n.
- However, in this text, what we are proposing is to do *almost* this.
- Namely, we propose to design a system that only works for situations when we have a fixed number of inputs n.
- To design such a system, we do the following.
- First, in each of many different application areas, we collect a large number P of patterns $(x_1^{(i)}, \ldots, x^{(i)}, y^{(i)})$, $i = 1, \ldots, P$ corresponding to this particular area.
- In most application areas, this is already done, we already have many such example.

17. Resulting proposal (cont-d)

- Then, for each application area, many times:
  - we select $K+1$ of these patterns $i_1, \ldots, i_K, i_{K+1}$ and
  - we use, as X, the first K of these patterns and the inputs corresponding to the $(K+1)$-st pattern:

$$X = (x_1^{(i_1)}, \ldots, x_n^{(i_1)}, y^{(i_1)}, \ldots, x_1^{(i_K)}, \ldots, x_n^{(i_K)}, y^{(i_K)}, x_1^{(i_{K+1})}, \ldots, x_n^{(i_{K+1})}).$$

- As the value $y = F(X)$ correspond to this tuple X, we use the value y for the $(K+1)$-st pattern, i.e., $y = y^{(i_{k+1})}$.
- Finally:
  - we use all the resulting pairs (X, y) – corresponding to different application areas and to different selection of $K+1$ patterns within each area
  - to train a neural network that will produce, in all application areas, y based on X.

18. What we expect

- Once trained, this neural network will transform each tuple X of the above type into an appropriate value y.
- In other words, it will:
  - take a small number K patterns from some application area and an input $x_1, \ldots, x_n$, and
  - generate the output corresponding to what we expect in this particular application area.
- In other words, this system will do exactly what we wanted: produce reasonable answer after a small number K of examples.
- How fast will this system be?
- Training this neural network may take forever.

19. What we expect (cont-d)

- However, as we have mentioned earlier:
  - once this neural network is trained,
  - it will produce its results really fast.
- In other words:
  - not only will the resulting system learn from a small number of examples,
  - it will also produce its results practically immediately, just like we human do.
- Again, this is exactly what we wanted.

20. Speculative comment

- We were discussing how to design a neural network that can learn from few examples – and learn fast.
- But there are already networks that seems to be doing this – namely, modern large linguistic models like GPT3 and ChatGPT.
- This ability of such models is largely a mystery.
- So maybe the explanation of their success is that these models are, in effect, already implementing the above idea?
- Indeed:
  - in contrast to the usual neural network that is usually trained on examples from one application area,
  - these models are trained on all kids of language examples, i.e., examples from all possible application areas,
  - and, as we have argued, such training is one of the main features that can lead to such training-fast-on-few examples.

21. Acknowledgments

This work was supported in part by:

- National Science Foundation grants 1623190, HRD-1834620, HRD-2034030, and EAR-2225395;
- AT&T Fellowship in Information Technology;
- program of the development of the Scientific-Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478, and
- a grant from the Hungarian National Research, Development and Innovation Office (NRDI).