Why Deep Learning Is More Efficient than Support Vector Machines, and How It Is Related to Sparsity Techniques in Signal Processing

Laxman Bokati[1], Vladik Kreinovich[1], Olga Kosheleva[1], and Anibal Sosa[2]
[1]University of Texas at El Paso, USA
lbokati@miners.utep.edu, olgak@utep.edu, vladik@utep.edu
[2]Universidad Icesi, Cali, Colombia, hannibals76@gmail.com

1. Main Objectives of Science and Engineering

- We want to make our lives better, we want to select actions and designs that will make us happier.
- We want to improve the world so as to increase our happiness level.
- To do that, we need to know:
  - what is the current state of the world, and
  - what changes will occur if we perform different actions.
- Crudely speaking:
  - learning the state of the world and learning what changes will happen is science, while
  - using this knowledge to come up with the best actions and best designs is engineering.

2. Need for Machine Learning

- In some cases, we already know how the world operates.
- E.g., we know that the movement of the celestial bodies is well described by Newton's equations.
- It is described so well that we can predict, e.g., Solar eclipses centuries ahead.
- In many other cases, however, we do not have such a good knowledge.
- We need to extract the corresponding laws of nature from the observations.

3. Need for Machine Learning (cont-d)

- In general, prediction means that:
  - we can predict the future value y of the physical quantity of interest
  - based on the current and past values $x_1, \ldots, x_n$ of related quantities.
- To be able to do that, we need to have an algorithm that:
  - given the values $x_1, \ldots, x_n$,
  - computes a reasonable estimate for the desired future value y.

4. Need for Machine Learning (cont-d)

- In the past, designing such algorithms was done by geniuses:
  - Newton described how to predict the motion of celestial bodies,
  - Einstein provided more accurate algorithms,
  - Schroedinger, in effect, described how to predict probabilities of different quantum states, etc.
- This still largely remains the domain of geniuses, Nobel prizes are awarded every year for these discoveries.
- However, now that the computers has become very efficient, they are often used to help.

5. Need for Machine Learning (cont-d)

- This use of computers is known as *machine learning*:
  - we know, in several cases $c = 1, \ldots, C$, which values $y^{(c)}$ corresponded to appropriate values $x_1^{(c)}, \ldots, x_n^{(c)}$;
  - we want to find an algorithm $f(x_1, \ldots, x_n)$ for which, for all these cases c, we have $y^{(c)} \approx f(x_1^{(c)}, \ldots, x_n^{(c)})$.
- The value y may be tomorrow's temperature in a given area.
- It may be a binary (0-1) variable deciding, e.g., whether a given email is legitimate or a spam.

6. Machine Learning: a Brief History

- One of the first successful general machine learning techniques was the technique of *neural networks*.
- In this technique, we look for algorithms of the type

$$f(x_1, \ldots, x_n) = \sum_{k=1}^{K} W_k \cdot s\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right) - W_0.$$

- Here, a non-linear function $s(z)$ is called an *activation function*, and values w_{ki} and W_k knows as *weights*.
- As the function $s(z)$, researchers usually selected the so-called *sigmoid function*

$$s(z) = \frac{1}{1 + \exp(-z)}.$$

- This algorithm emulates a 3-layer network of biological neurons – the main brain cells doing data processing.

7. Machine Learning: a Brief History (cont-d)

- In the first layer, we have input neurons that read the inputs $x_1, \ldots, x_n$.
- In the second layer – called a *hidden layer* – we have K neurons each of which:
  - first generates a linear combination of the input signals: $z_k = \sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}$
  - and then applies an appropriate nonlinear function $s(z)$ to z_k, resulting in a signal $y_k = s(z_k)$.
- The processing by biological neurons is well described by the sigmoid activation function.
- This is the reason why this function was selected for artificial neural networks in the first place.
- After that, in the final output layer, the signals y_k from the neurons in the hidden layer are combined.

8. Machine Learning: a Brief History (cont-d)

- The linear combination $\sum_{k=1}^{K} W_k \cdot y_k - W_0$ is returned as the output.
- A special efficient algorithm – *backpropagation* – was developed to *train* the corresponding neural network.
- This algorithm finds the values of the weights that provide the best fit for the observation results $x_1^{(c)}, \ldots, x_n^{(c)}, y^{(c)}$.

9. Support Vector Machines (SVM): in Brief

- Later, in many practical problem, a different technique became more efficient: the SVM technique.
- Let us explain this technique on the example of a *binary classification* problem, i.e., a problem in which:
  - we need to classify all objects (or events) into one of two classes,
  - based on the values $x_1, \ldots, x_n$ of the corresponding parameters
- In such problems, the desired output y has only two possible values; this means that:
  - the set of all possible values of the tuple $x = (x_1, \ldots, x_n)$
  - is divided into two non-intersecting sets S_1 and S_2 corresponding to each of the two classes.

10. Support Vector Machines (cont-d)

- We can thus come up with a continuous f-n $f(x_1, \ldots, x_n)$ such that $f(x) \ge 0$ for $x \in S_1$ and $f(x) \le 0$ for $x \in S_2$.
- As an example of such a function, we can take $f(x) = d(x, S_2) - d(x, S_1)$, where $d(x, S) \stackrel{\text{def}}{=} \inf\limits_{s \in S} d(x, s)$.
- If $x \in S$, then $d(x, s) = 0$ for $s = x$ thus $d(x, S) = 0$.
- For points $x \in S_1$, we have $d(x, S_1) = 0$ but usually $d(x, S_2) > 0$, thus $f(x) = d(x, S_2) - d(x, S_1) > 0$.
- For points $x \in S_2$, we have $d(x, S_2) = 0$ while, in general, $d(x, S_1) > 0$, thus $f(x) = d(x, S_2) - d(x, S_1) < 0$.
- In some cases, there exists a linear function that separates the classes: $f(x_1, \ldots, x_n) = a_0 + \sum\limits_{i=1}^{n} a_i \cdot x_i$.
- In this case, there exist efficient algorithms for finding the corresponding coefficients a_i.

11. Support Vector Machines (cont-d)

- For example, we can use linear programming to find the values a_i for which:
  - $a_0 + \sum\limits_{i=1}^{n} a_i \cdot x_i > 0$ for all known tuples $x \in S_1$, and
  - $a_0 + \sum\limits_{i=1}^{n} a_i \cdot x_i < 0$ for all known tuples $x \in S_2$.
- In many practical situations, however, such a linear separation is not possible.
- In such situations, we can take into account the known fact that:
  - any continuous function on a bounded domain
  - can be approximated, with any given accuracy, by a polynomial.

12. Support Vector Machines (cont-d)

- Thus, we can separate the classes by checking whether the f-approximating polynomial is > 0 or < 0:

$$P_f(x) = a_0 + \sum_{i=1}^n a_i \cdot x_i + \sum_{i=1}^n \sum_{j=1}^n a_{ij} \cdot x_i \cdot x_j + \ldots.$$

- We can map each original n-dimensional point $x = (x_1, \ldots, x_n)$ into a higher-dimensional point

$$X = (X_1, \ldots, X_n, X_{11}, X_{12}, \ldots, X_{nn}, \ldots) =$$
$$(x_1, \ldots, x_n, x_1^2, x_1 \cdot x_2, \ldots, x_n^2, \ldots).$$

- Then in this higher-dimensional space, the separating function becomes linear:

$$P_f(X) = a_0 + \sum_{i=1}^n a_i \cdot X_i + \sum_{i=1}^n \sum_{j=1}^n a_{ij} \cdot X_{ij} + \ldots.$$

- And we know how to effectively find a linear separation.

13. Support Vector Machines (cont-d)

- Instead of polynomials, we can use another basis $e_1(x)$, $e_2(x)$, …, to approximate a separating function as

$$a_1 \cdot e_1(x) + a_2 \cdot e_2(x) + \ldots$$

14. Where the Term SVM Comes From

- The name of this technique comes from the fact that:
  - when solving the corresponding linear programming problem,
  - we can safely ignore many of the samples and
  - concentrate only on the vectors X which are close to the boundary between the two sets.
- If we get linear separation for such *support vectors*, we will automatically get separation for other vectors X.
- This possibility to decrease the number of iterations enables us:
  - to come up with algorithms for the SVM approach
  - which are more efficient than general linear programming algorithms.

15. Deep Learning: a Brief Description

- Lately, the most efficient machine learning tool is *deep learning*.
- Deep learning is a version of a neural network.
- The main difference is that:
  - instead of a large number of neurons in a hidden layer,
  - we have multiple layers with a relatively small number of neurons in each of them.
- Similarly to the traditional neural networks, we start with the inputs $x_1, \ldots, x_n$.
- These values are inputs $x_i^{(0)}$ to the neurons in the 1st layer.

16. Deep Learning: a Brief Description (cont-d)

- On each layer k, each neuron takes, as inputs, outputs $x_i^{(k-1)}$ from the previous layer and returns the value

$$x_j^{(k)} = s_k\left(\sum_i w_{ij}^{(k)} \cdot x_i^{(k-1)}\right) - w_{0j}^{(k)}.$$

- For most layers, instead of the sigmoid, it turns out to be more efficient to use a piece-wise linear function

$$s_k(x) = \max(x, 0).$$

- In the last layer, sometimes, the sigmoid is used.
- There are also layers in which inputs are divided into groups, and:
  - we combine inputs from each group into a single value,
  - e.g., by taking the max of the inputs.

17. Deep Learning: a Brief Description (cont-d)

- In addition to backpropagation, several other techniques are used to speed up computations.
- E.g., instead of using *all* the neurons in training, one of the techniques is:
  - to only use, on each iteration, *some* of the neurons and then
  - combine the results by applying an appropriate combination function (geometric mean).

18. Natural Questions

- So far, we have described what happened:
  - support vector machines turned out to be more efficient in machine learning, and
  - deep learning is, in general, more efficient than support vector machines.
- A natural question is: why?
- How can we theoretically explain these facts – thus increasing our trust in these conclusions?

19. What We Do in This Talk

- In our previous papers, we explained why deep learning is more efficient than the traditional neural networks.
- We also explained:
  - the selection of piece-wise linear activation functions,
  - why some combination functions are more efficient,
  - and several other features of deep learning.
- In this talk, we extend these explanations to the comparison between SVM and neural networks.
- The resulting explanation:
  - will help us understand yet another empirical fact,
  - the empirical efficiency of sparse techniques in signal processing.

20. Support Vector Machines Vs. Neural Networks

- This empirical comparison is the easiest to explain.
- To train a traditional neural network, we need to find the weights W_k and w_{ki} for which

$$y^{(c)} \approx \sum_{k=1}^{K} W_k \cdot s\left(\sum_{i=1}^{n} w_{ki} \cdot x_i^{(c)} - w_{k0}\right) - W_0.$$

- Here, the activation function $s(z)$ is non-linear.
- So we have a system of non-linear equations for finding the corresponding weights W_k and w_{ki}.
- In general, solving a system of nonlinear equations is NP-hard even for quadratic equations.

21. SVM Vs. Neural Networks (cont-d)

- In contrast, for support vector machines:
  - to find the corresponding coefficients a_i,
  - it is sufficient to solve a linear programming problem.
- This can be done in feasible time.
- This explains why support vector machines are more efficient than traditional neural networks.

22. Support Vector Machines Vs. Deep Learning

- At first glance, the above explanation should work for the comparison between SVM and deep networks:
  - in the first case, we have a feasible algorithm, while
  - in the second case, we have an NP-hard problem that may require very long (exponential) time.
- However, this is only at first glance.
- The above comparison assumes that:
  - all the inputs $x_1, \ldots, x_n$ are independent,
  - i.e., that none of them can be described in terms of one another.
- In reality, most inputs are dependent in this sense.

23. SVM Vs. Deep Learning (cont-d)

- This is especially clear in many engineering and scientific applications, where:
  - we use the results of measuring appropriate quantities at different moments of time as inputs,
  - but we know that these quantities are usually *not* independent,
  - they satisfy some differential equations.
- As a result, we do not need to use all n inputs.
- If there are $m \ll n$ independent ones, this means that:
  - it is sufficient to use only m of the inputs – or, alternatively, m different combinations of inputs,
  - as long as they combinations are independent (and, in general, they are).

24. SVM Vs. Deep Learning (cont-d)

- And this is exactly what is happening in a deep neural network.
- Indeed, in the traditional neural network:
  - we can have many neurons in the processing (hidden) layer,
  - so we can have as many neurons as inputs (or even more).
- In contrast, in the deep neural networks, the number of neurons in each layer is limited.
- In particular:
  - the number of neurons in the first processing layer
  - is, in general, much smaller than the number of inputs.

25. SVM Vs. Deep Learning (cont-d)

- And all the resulting computations are based *only* on the outputs $x_k^{(1)}$ of the neurons from this first layer.
- Thus, in effect, the desired quantity y is computed
  - not based on all n inputs, but
  - based only on m combinations – where m is the number of neurons in the first processing layer.
- In spite of this limitation:
  - deep neural networks seem to provide a universal approximation
  - to all kinds of actual dependencies.
- This is an indication that inputs are usually dependent on each other.

26. SVM Vs. Deep Learning (cont-d)

- This dependence explains why, empirically, deep neural networks work better than support vector machines:
  - deep networks implicitly take into account this dependency, while
  - support vector machines do not take any advantage of this dependency.
- As a result, deep networks need fewer parameters than would be needed for n independent inputs.
- Hence, during the same time, they can perform more processing and thus, get more accurate predictions.

27. What Are Sparsity Techniques

- The above explanations help us explain another empirical fact – that:
  - in many cases of signal and image processing,
  - sparsity techniques has been very effective.
- Usually, in signal processing:
  - we represent the signal $x(t)$
  - by the coefficients a_i of its expansion in the appropriate basis $e_1(t)$, $e_2(t)$, etc.:

$$x(t) \approx \sum_{i=1}^{n} a_i \cdot e_i(t).$$

- In Fourier analysis, we use the basic of sines and cosines.
- In wavelet analysis, we use wavelets as the basis, etc.

28. Sparsity Techniques (cont-d)

- Similarly, in image processing, we represent an image $I(x)$ by the coefficients of its expansion over some basis.
- It turns out that in many practical problems, we can select the basis $e_i(t)$ in such a way that:
  - for most actual signals,
  - the corresponding representation becomes *sparse*
  - in the sense that most of the corresponding coefficients a_i are zeros.
- This phenomenon leads to very efficient algorithms for signal and image processing; however:
  - while empirically successful,
  - from the theoretical viewpoint, this phenomenon largely remains a mystery:
  - why can we find such a basis?

29. Our Explanation

- The shape of the actual signal $x(t)$ depends on many different phenomena.
- So, in general, we can say that $x(t) = F(t, c_1, \ldots, c_N)$ for some function F, where c_iN are parameters.
- Usual signal processing algorithms implicitly assume that we can have all possible combinations of c_i's.
- However, as we have mentioned, in reality, the corresponding phenomena are dependent on each other.
- As a result, there is a functional dependence between the corresponding values c_i.
- Only few of them $m \ll N$ are truly independent, others can be determined based on the these few ones.

30. Our Explanation (cont-d)

- If we denote the corresponding m independent values by $b_1, \ldots, b_m$, then the above description takes the form

$$x_i(t) = G(t, b_1, \ldots, b_m) \text{ for some } G.$$

- It is known that any continuous function can be approximated by piecewise linear functions.
- If we use this approximation instead of the original function G, then we conclude that:
  - the domain of possible values of the tuples $(b_1, \ldots, b_m)$ is divided into a small number of sub-domains $D_1, \ldots, D_p$
  - on each of which D_j the dependence of $x_i(t)$ on the values b_i is linear:

$$x_i(t) = \sum_{k=1}^{m} b_k \cdot e_{jk}(t) \text{ for some f-ns } e_{jk}(t).$$

31. Our Explanation (cont-d)

- Let us take all $m \cdot p$ the functions $e_{jk}(t)$ corresponding to different subdomains as the basis.
- Then, we conclude that:
  - on each subdomain, each signal can be described by no more than $m \ll p \cdot m$ non-zero coefficients,
  - this is exactly the phenomenon that we observe and utilize in sparsity techniques.

32. Acknowledgments

This work was supported in part by the National Science Foundation via grants:

- 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science),
- and HRD-1242122 (Cyber-ShARE Center of Excellence).

The authors are thankful to Laszlo Koczy for his encouragement.