The “Psychological” Limits of Neural Computation

RAZVAN ANDONIE
Department of Electronics and Computers,
Transylvania University, 2200 Brasov, Romania

Abstract: Recent results changed essentially our view concerning the generality of neural networks’ models. Presently, we know that such models (i) are more powerful than Turing machines if they have an infinite number of neurons, (ii) are universal approximators, (iii) can represent any logical function, (iv) can solve efficiently instances of NP-complete problems. In a previous paper [1], we discussed the computational capabilities of artificial neural networks vis-a-vis the assumptions of classical computability. We continue here this discussion, concentrating on the worst case “psychological” limits of neural computation. Meanwhile, we state some open problems and presumptions concerning the representation of logical functions and circuits by neural networks.

1. Neural networks and Turing machines

It is known that each algorithm is Turing solvable. In the context of function computability, the Church-Turing thesis states that each intuitively computable function is Turing computable. The languages accepted by Turing machines form the recursively enumerable language family $L_0$ and, according to the Church-Turing thesis, $L_0$ is also the class of algorithmic computable sets. In spite of its generality, the Turing model cannot solve any problem. Recall, for example, that the halting problem is Turing unsolvable: it is algorithmic undecidable if an arbitrary Turing machine will eventually halt when given some specified, but arbitrary, input.

McCulloch and Pitts [28] asserted that neural networks are computationally universal. A neural network implementation of a Turing machine was provided by Franklin and Garzon [13, 14]. The consequence is, of course, that any algorithmic problem that is Turing solvable can be encoded as a problem solvable by a neural network: neural networks are at least as powerful as Turing machines.

The converse has been widely presumed true [18], since computability has become synonymous with Turing computability. Nonetheless, Franklin and Garzon [13, 14] proved that the halting problem, when suitably encoded on a neural network, is solvable by an infinite neural network. Despite its appearance, this result is not a counterexample to the Church-Turing thesis since the thesis concerns only algorithmic problems solvable by finitary means. Intuitively, it is not a surprise that a device with an infinite number of processing units is more complex than a Turing machine. The infinite time complexity of the halting problem on a Turing machine has been transferred into the infinite number of neurons. The consequence of this property has a fundamental importance: infinite neural networks are more powerful than Turing machines.

2. Function approximation

The approximation (or prediction) of functions that are known only at a certain number of discrete points is a classical application of multilayered neural networks. Of fundamental importance was the discovery [20] that a classical mathematical result of Kolmogorov (1957) was actually a statement that for any continuous mapping \( f : [0,1]^n \subset \mathbb{R}^n \rightarrow \mathbb{R}^m \) there must exist a three-layered feedforward neural network of continuous type neurons (having an input layer with \( n \) neurons, a hidden layer with \((2n+1)\) neurons, and an output layer with \( m \) neurons) that implements \( f \) exactly. This existence result was the first step. Cybenko [10] showed that any continuous function defined on a compact subset of \( \mathbb{R}^n \) can be approximated to any desired degree of accuracy by a feedforward neural network with one hidden layer using sigmoidal nonlinearities. Many other papers have investigated the approximation capability of three-layered networks in various ways (see [37]). More recently, Chen et al. [8] pointed out that the boundedness of the sigmoidal function plays an essential role for its being an activation function in the hidden layer, i.e., instead of continuity or monotony, the boundedness of sigmoidal functions ensures the network’s approximation capability of functions defined on compact sets in \( \mathbb{R}^n \).

In addition to sigmoid functions, many others can be used as activation functions of universal approximator feedforward networks [9]. Girosi and Poggio [17] proved that radial basis function networks also have universal approximation property. Consequently, a feedforward neural network with a single hidden layer has sufficient flexibility to approximate with a given error any continuous function defined on a compact (these conditions may be relaxed). However, there are at least three important limitations:

a) These existence proofs are rather formal and do not guarantee the existence of a reasonable representation in practice. In contrast, a constructive proof was given [27] for networks with two hidden layers. An explicit numerical implementation of the neural representation of continuous functions was recently discovered by Sprecher [35,36].

b) The arbitrary accuracy by which such networks are able to approximate any quite general function is based on the assumption that arbitrary large parameters (weights and biases) and enough hidden units are available. However, in practical situations, both the size of parameters and the numbers of hidden neurons are bounded. The problem of how the universal approximation property can be achieved constraining the size of parameters and the number of hidden units was recently examined by Kurková [26].

c) These “universal approximation” proofs are commonly used to justify the notion that neural networks can “do anything” (in the domain of function approximation). What is not considered by this proofs is that networks are simulated on computers with finite accuracy. Wray and Green [38] showed that approximation theory results cannot be used blindly without consideration of numerical accuracy limits, and that these limitations constrain the approximation ability of neural networks.
The relationship between networks with one hidden layer and networks with several hidden layers is not yet well understood. Although one hidden layer is always enough, in solving particular problems it is often essential to have more hidden layers. This is because for many problems an approximation with one hidden layer would require an impractically large number of hidden neurons, whereas an adequate solution can be obtained with a tractable network size by using more than one hidden layer.

3. Representation of logical functions using neural networks

A finite mapping is defined as a mapping from a finite subset of the Euclidian space $\mathbb{R}^n$ into $\mathbb{R}$. Ito [21] proved that feedforward neural networks with a single hidden layer and with any sigmoidal activation function can implement an arbitrary finite mapping exactly. In the special case where the finite mapping is a logical function $f : \{0, 1\}^n \rightarrow \{0, 1\}$, the proof is straightforward if we consider an adequate discrete neural network with $2^n$ neurons in the hidden layer. (McCulloch and Pitts have proved in 1943 that any logical function can be implemented by a neural network [28].) This representation corresponds to a disjunctive normal form (DNF) of the function, with one OR gate (the neuron from the output layer) and a maximum of $2^n$ AND gates (the hidden neurons). Each hidden neuron corresponds to exactly one term of the DNF expression. A term is an ANDing together of a finite number of Boolean variables, any of which may be negated. We shall refer CNN (Canonical Neural Network) to this discrete neural representation of a logical function.

The CNN is not the only possible network architecture for representing a logical function. A single discrete neuron is capable of realizing not only the universal NAND (or NOR) function, but all linear threshold functions. Therefore, a single $n$-input discrete neuron is much more powerful than a single $n$-input NAND or NOR gate and it is to be expected that a more general neural net representation of a logical function may require a smaller number of hidden neurons than the CNN representation. In fact, this can be proved (see [19]). For instance, the XOR function with two arguments can be represented with 2, not 4, hidden neurons. Moreover, even a representation with 2 neurons is possible, in an arbitrary interconnected network with no feedback [19].

One efficient way to describe the topology and the weights of a feedforward neural network would be to list for each hidden or output neuron $q_i$ the indices $(j)$ and corresponding weights $(w_{ij})$ for every neuron $q_j$ connected to $q_i$. This way of describing a feedforward neural network will be called a list expression of the network.

The very first question to be answered is whether, for a logical function, it has any sense to use a neural network representation instead of the classical DNF representation. Considering the evaluation speed and the space requirements, there are clear advantages of neural representations (as list expressions) over DNF
representations [see 15]. For example, consider the majority function on \( k \) inputs. Such a function can be represented as a feedforward neural network, described by a list expression using \( O(k) \) symbols. The DNF representation requires, however, \( C(k, \lceil (k + 1) / 2 \rceil) \) terms. Therefore, the number of terms for the DNF representation of the majority function grows exponentially in \( k \).

It is relatively easy to establish an upper bound on the size of a feedforward net for realizing arbitrary, even partially specified, logical functions. Considering a partially specified logical function, defined on a set of \( m \) arbitrary points in \( \{0, 1\}^n \), with \( m \leq 2^n \), then, a single hidden layer discrete neural network with \( m \) hidden neurons and one output neuron is sufficient to realize the function [19]. It is perhaps more interesting to know the lower bounds on the number of hidden neurons for realizing any logical function by a discrete neural net with a single output neuron. Hassoun [19] proved the existence of the following lower bounds, in the limit of large \( n \):

1. A feedforward net with one hidden layer requires more than \( m / [n \log_2 (m e / n)] \) hidden neurons to represent any function \( f: \mathbb{R}^n \to \{0, 1\} \) defined on \( m \) arbitrary points in \( \mathbb{R}^n, \ m \geq 3n \).
2. A two-hidden-layer feedforward net having \( k / 2 \) (\( k \) is even) neurons in each of its hidden layers requires that \( k \geq 2(m / \log_2 m)^{1/2} \) to represent any function \( f: \mathbb{R}^n \to \{0, 1\} \) defined on \( m \) arbitrary points in \( \mathbb{R}^n, \ m \gg n^2 \).
3. An arbitrary interconnected network with no feedback needs more than \((2m / \log_2 m)^{1/2}\) neurons to represent any function \( f: \mathbb{R}^n \to \{0, 1\} \) defined on \( m \) arbitrary points in \( \mathbb{R}^n, \ m \gg n^2 \).

We can conclude that *when the training sequence is much larger than \( n \), networks with two or more hidden layers may require substantially fewer units than single-hidden-layer-networks*. Other bounds on the number of units for computing arbitrary logical functions by multilayer feedforward neural networks may be found in [32].

The fact that *neural networks can implement any logical function* needs more attention. At first sight, such a representation seems useless in practice, because the number of required neurons may be very large. There are three important observations:

a) The representation of arbitrary logical functions is a NP-complete problem, which here finds its expression in the fact that in general an exponentially growing number of hidden neurons might be required.

b) For individual logical functions, the number of hidden neurons can be much smaller. For example, symmetric functions with \( p \) inputs are representable by networks with a linear (with respect to \( p \)) number of neurons in the hidden layer [15]. *Symmetric functions* are an important class of logical functions giving outputs that are not changed if inputs are permuted. Special cases of symmetric functions are: AND function, OR function, XOR function, majority function.
c) A more general neural network, representing an individual function, may require substantial fewer neurons than the straightforward CNN representation of this function.

A very practical problem arises: How to minimize the neural network representation of a particular logical function (that means, how to obtain an implementation with a minimum number of neurons). For CNN representations, the problem is not new and is equivalent to the minimization of a DNF logical function. Therefore, one might apply a usual minimization algorithm (Veitch-Karnaugh or Quine-McCluskey [37]). The minimization of a DNF is a NP-complete problem (see [16], problem L09). It is not surprising that the minimization of logical functions having more than two layers is also a hard problem [37].

We have to conclude that the minimization of a neural network representation of a logical function is a hard problem even for one hidden layer feedforward architectures.

Another important result concerns the complexity theory of decision problems. A decision problem is based on the logical functions conjunction, disjunction, and complement. If we enlarge the family of decision problems, including decision problems based on linear threshold functions, we obtain new complexity classes [31].

4. The complexity of learning in neural networks

First, we shall consider the general problem of supervised learning in multilayered feedforward neural networks. By “learning” we understand here memorization: the network is taught to remember (to represent) some given data. Arising immediately is the question whether there exists an efficient algorithm for solving this learning problem. How does learning time scale up with network size (number of neurons and connections) and data size? Judd proved that the learning problem in its general form is NP-complete (learning in a network with a single hidden layer is shown to be equivalent to an algorithm for solving SAT3). The learning problem is NP-complete even for networks with a single hidden layer. These results are almost entirely independent of the type of activation function. Blum and Rivest [6], and, more recently, DasGupta et al. [11] and Šima [39] showed similar NP-completeness results. In particular, this implies that backpropagation is (generally) NP-hard [39].

However, there are fast learning algorithms for cases where the network is of a very restricted design, or where the data to be memorized are very simple. It would be interesting to know if there are some useful classes of polynomial learnable tasks. Polynomial learning capability has been proved for some classes of problems onto certain carefully selected architectures of neural networks [22], [15], [19]. Judd’s conclusion is (we quote): “As time passes, definitions will get sharper and more focused on exactly those classes of functions that are learnable with reasonable neural networks, and it will be clearer what our engineering ambitious should rightly be” [23]. Minsky has a similar opinion (we quote again): “As the field of connectionism becomes more mature, the quest for a general solution to all learning problems will involve into an understanding of which types of learning processes are likely to work on which classes of problems” [30].
It has been an empirical observation that some algorithms (notably backpropagation) work well on nets that have only a few hidden layers. One might be tempted to infer that shallow nets would be intrinsically easier to teach. Judd’s result shows that this is not the case: learning in a network with one hidden layer is (generally) NP-complete.

5. Learning logical functions

In machine learning it is known [24] that while the most general classes of logical functions are not learnable in polynomial time, a wide variety of restricted classes of functions are. There is still a large gap between these positive and negative learnability results, and lying in this gap are some of the most natural classes of functions. Principal among these is the DNF of an arbitrary function. Although the learnability of general DNF remains open, there are polynomial time algorithms for learning various restricted subclasses of DNF [2].

We shall discuss now the following problem. Is it possible to obtain efficient neural representations of logical functions by using some standard neural learning techniques? Such a strategy was proposed, for instance, by Mézard and Nadal [29]. It is based on the perceptron learning algorithm and can be applied if many additional hidden layers are allowed. The last limitation makes this strategy inconvenient for our purpose.

There are at least two types of questions that one may ask, both closely related. The first question is whether an arbitrary logical function of \( n \) arguments can be learned by a neural network in an amount of time that only grows polynomially with \( n \). This learning can be also considered from the point of view of “weak learnability” (PAC learning), as defined by Valiant (see [24]). Considering Judd’s result, the problem of learning an arbitrary function by an arbitrary feedforward neural network is NP-complete. From a practical point of view, this says nothing; it is only stated that we are not able to learn efficiently an arbitrary function on an arbitrary network. Therefore, we shall reformulate our question: is there a neural network, from a semnificative class of networks, on which an arbitrary logical function can be learned efficiently? Very little is known in this direction and the mathematical problem is far from trivial (see [34]). Things become even more practical if we consider specific logical functions, not “arbitrary functions”.

The second question is whether a logical function can be learned (in any amount of time) from only a few examples of the pair (function argument, function value). This is commonly called the problem of generalization. Several authors tried to answer this question in the context of neural networks. For example, using simulated annealing to train the network, Carnevali and Paternello [7] checked how many examples were needed for a neural network to learn a particular logical function.

6. The optimization of circuits

To make the discussion clearer, let us remember that:
a) It seems advantageous to use neural network representations of logical functions.

b) The neural network representation of a logical function might be expensive (considering the number of neurons).

c) Learning neural network representations of logical functions may be computationally expensive.

d) A neural network representation of some logical functions may be obtained efficiently by generalization, memorizing examples for the pair (function argument, function value).

These properties might be obviously extended over the wider class of circuits, which are finite combinations of logical functions.

The size of a circuit equals the number of its gates. The depth of a circuit is the number of gates on its longest path. A circuit is efficient if size and depth are small. Our knowledge about circuits is far from that required to always design efficient circuits. We have no general methods to simultaneously minimize the size and the depth of a circuit as a trade-off result [37]. The independent minimization of the two parameters may lead to expensive solutions. For instance, a minimal DNF representation of a logical function does not necessarily lead to an efficient circuit, in spite of the fact that we minimized first the depth (restricting the problem to the class of two-level circuits) and then the number of gates. This becomes more evident in the case of the \( n \)-parity logical function. Another important parameter of a circuit is the fan-in (the number of incoming edges) of each gate. For VLSI engineers, the area (\( A \)) and the delay of the circuit (\( T \)) are standard optimization costs, and a circuit is considered VLSI-optimal if it minimises the combined measure \( AT^2 \).

For practical reasons, it would be interesting to find a wide subclass of circuits that can be efficiently learned and represented by neural networks. Meanwhile, this could also be a solution for obtaining “optimal” circuits. Thus, one could hope to obtain circuits with small size and depth, or VLSI-optimal circuits. There are two main approaches for building efficient neural networks’ representations: learning techniques (for instance, variants of backpropagation), and constructive (direct design) techniques (see [33]). The research in this area is presently very active.

As a first step, the following two experiments were performed by one of my students, A. Cataron:

**Experiment A.** A two bits adder with carry was represented as a combination of two DNF logical functions. It was minimized using the Veitch-Karnaugh method, leading to a quite complicated structure with 12 gates. Implemented as a CNN, the circuit required 3 entries, 7 hidden neurons and 2 output neurons. Next, a neural network with a single hidden layer was “taught” by backpropagation to “memorize” the circuit. The number of hidden neurons was reduced as much as possible (so that learning by backpropagation was still effective). The resulted network had only 2 hidden neurons and 2 output neurons. Thus, the most compact neural representation of the circuit was obtained.

The Adder Problem is one of the many test problems often used to benchmark neural network architectures and learning techniques, since it is sufficient complex. Keating and Noonan [25] obtained the neural network representation of a three bits
adder, using simulated annealing to train the network. The resulting neural representation of the adder did not compare well with the minimum circuit obtained by the Veitch-Karnaugh method. This happened, in our opinion, because their method does not minimize the size of the neural net; the architecture of the neural net remains unchanged during and after the training process. Incremental (architecture-changing) learning techniques seems to be more efficient for this problem.

**Experiment B.** A more complex circuit, a priority encoder with 29 gates, was too large to be implemented as a CNN. Again, a neural network with a single hidden layer was “taught” by backpropagation to “memorize” the circuit. The “teaching” sequence was, as before, the entire set of 512 associations. The result was a neural network representation with 9 entries, 4 hidden neurons and 5 output neurons. Since the teaching sequence was too large (and time consuming), several learning tests with incomplete teaching sequences were performed. The results are encouraging, even if only approximations of the initial functions have been obtained (this means that the generalization process was not perfect).

Complex logical functions are difficult to represent as CNN because of the constructive procedure we used. Instead, learning neural network representations on multilayered feedforward networks seems to be more practical. Learning and hidden layer minimization were performed in a heuristic way. Beside heuristics, another technique may help make progress: generalization. It is interesting to find some special logical circuit for which approximately correct generalization is satisfactory.

Obtaining the neural network representations of a circuit does not mean that we obtained the circuit itself; the hardware implementation must follow. An immediate application would be to directly map the neural network representation in FPGAs [4]. It is not straightforward to compare the generally accepted minimum circuits (the adder and the priority encoder) with the neural obtained ones. The hardware implementation of a neural network is itself a challenging task with many open problems. For instance, Beiu [3] proved that digital VLSI-optimal implementations of neural networks could be done by constant fan-in digital circuits. The standard procedure in learning a logical function by backpropagation is to perform learning on a sigmoid neural network and to replace afterthat the continuous neurons with discrete neurons. Sometimes it might be interesting to implement directly in hardware the resulting continuous neural network. Fan-in dependent depth-size trade-offs were observed when trying to implement sigmoid type neural networks [5].

7. Final remarks

Our motivation for writing this paper has been given by the new results obtained during the last few years in neural computation. Some of these results make us reflect (and doubt?), once again, on the limits of computability. Although, at a closer look, we re-discover some “old” limits:
i) Infinite neural networks are more powerful than Turing machines, but this result is not a counterexample to the Church-Turing thesis.

ii) Neural networks are universal approximators but there are serious limitations.

iii) Neural networks can represent any logical function but it is quite difficult to find efficient neural representations for complex logical functions (or circuits).

iv) Neural networks are frequently used to solve instances of NP-complete optimization problems. This does not prove that $P = NP$. The neural learning problem of a logical function gives an insight about how NP-complete problems can be “solved” by neural computing.

The infinite family of circuits represents the advance of technology over time. It is likely that in the next ten years we will produce a family of circuits scaling from current technological limits up to brain-scale computations. Nanotechnology (i.e., molecular scale engineering) will dominate 21st century economics. However, building such “Avogadro Machines” (with a trillion trillion processing elements) will demand an enormous complexity of the circuits. The complexity of such circuits shall be so huge that, using the present techniques, we shall not be able to design them: it will be impossible to predict function from structure. One possible way to design a complex circuit is by learning it’s neural network representations (i.e., using the concept of “evolutionary hardware” [12]). Even if the learning procedure risks to become a hard problem, this method seems to be of interest for the future.

We discussed here the worst case “psychological” limits of neural computation. From this point of view, we have to be rather pessimistic, accepting the late observations of Minsky and Papert [30]. Little good can come from statements like “learning arbitrary logical functions is NP-complete”. We have to quote here Minsky and Papert [30]:

“It makes no sense to seek the “best” network architecture or learning procedure because it makes no sense to say that any network is efficient by itself: that makes sense only in the context of some class of problems to be solved”.

From the point of view of restricted classes of problems which can be solved efficiently on neural networks we have to be optimistic. The “psychological” limits exist only in our mind. In practice, we have to solve particular, not general, problems.

Acknowledgement: I am grateful to Valeriu Beiu for many valuable comments that helped me to complete this paper.
References:
26. Kurková V. Approximation of functions by perceptron networks with bounded number of hidden units. Neural Networks 1995; 8: 745-750
32. Paugam-Moisy H. Optimisations des réseaux de neurones artificiels. These de doctorat, Ecole Normal Supérieure de Lyon, LIP-IMAG, URA CNRS nr. 1398, 1992
35. Sprecher DA. A numerical implementation of Kolmogorov’s superpositions. Neural Networks 1995; 8: 1-8