Information Theory, Complexity, and Neural Networks

Yaser S. Abu-Mostafa

OVER THE PAST FIVE OR SO YEARS, A NEW WAVE of research in neural networks has emerged. One of the areas that has attracted a number of researchers is the mathematical evaluation of neural networks as information processing systems. In this article, we discuss some of the main results in this area. Researchers have addressed, in specific terms, the questions of memory capacity, computing power, and learning capability of different neural network models. The performance of a conventional computer is usually measured by its speed and memory. For neural networks, measuring the computing performance requires new tools from information theory and computational complexity.

Neural network models offer an interesting alternative to performing certain computations. They have been considered, particularly, for unstructured computations, such as pattern recognition and artificial intelligence problems, and approximations to large optimization problems.

The performance of a conventional computer is usually measured by its speed and memory. For neural networks, measuring the computing performance requires new tools from information theory and computational complexity.

There are two popular models of neural networks; the feedback model [9] and the feed-forward model [15]. The feedback model is what triggered the current wave of interest in neural networks. The architecture of feedback networks can be described as an undirected graph (see Figure 1); often, the connections are bidirectional and symmetric in the models. The nodes are called neurons and the edges are called synapses.

What characterizes a neural architecture in general, whether it is feedback or feed-forward, is that the number of neurons is huge, and that each neuron performs a very simple task. In many, if not most, models, neurons perform threshold logic only. Another common characteristic of many neural networks is that the number of synapses per neuron is large. Usually a neuron is connected to a good fraction of all other neurons.

In the feed-forward model, the neurons are arranged into layers (see Figure 2). There are only directed synapses between each layer and the next. Thus, the connection is loop-free. The inputs are applied to the first layer, and the outputs are collected from the last layer. A feed-forward network is a special case of combinational circuits, with the additional feature that the intermediate variables in the network can assume non-binary values.

The neuron in both models performs the same function (as shown in Figure 3). The output y is determined by the inputs $x_1,...x_N$ according to a threshold rule. In the case of binary variables, the neuron simulates a function from $\{-1,+1\}^N$ to $\{-1,+1\}$ (we adopt a neural network notation taking the binary convention to be $+1$'s or -1's instead of 1's or 0's). The output depends on the input through a set of real numbers called the weights $w_1...w_N$, a weight for every input variable. If the sum

$$\sum_{i=1}^{N} w_i x_i \tag{1}$$

exceeds an internal threshold, t, the output y is set to $+1$; if it is less than t, y is set to -1.

The set of functions that can be implemented using a single neuron is well understood. It is the set of threshold functions, or linearly-separable functions. If we consider the hypercube $\{-1,+1\}^N$, any dichotomy that can be represented by a hyperplane that separates the points can be simulated by a single neuron. Non-binary neurons are also commonplace in neural network models. In this case, the threshold function produces an output that varies continuously between -1 and $+1$ as the signal

$$\sum_{i=1}^{N} w_i x_i - t \tag{2}$$

varies from large negative to large positive.

The operation of the feed-forward model is that of a combinational circuit, where the inputs propagate and interact in one direction to produce the output. The computation time is the time required for the signals to propagate and for the output to settle. The operation of the feedback model is closer to that of a sequential, or asynchronous, computer, where the system is initialized to a state and evolves in time to a final state. This simulates a computation, where the initial state is the input and the final state is the output.

0163-6804/89/0011-0025 $01.00 © 1989 IEEE

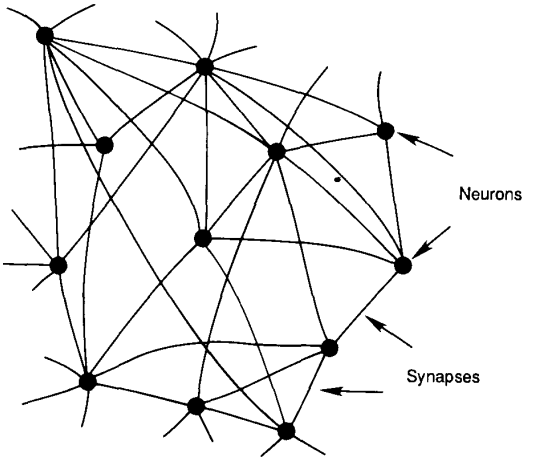

Fig. 1. Feedback network.

The question of stability does not arise in feed-forward networks because they are loop-free. It is important to predict how a feedback network evolves in time when the neurons are initialized to a certain state vector of bits. This evolution is analyzed with the help of an energy function that can be defined in terms of the states of the neurons and the (fixed) weights and thresholds. Under certain conditions, the energy decreases monotonically as the network moves from one state to the next. In these cases, stability and convergence can be addressed by analyzing the descent of the scalar energy function instead of the transition of the state vector. This parity of state vector transition and energy function descent is the key to understanding how to perform actual computations using feedback networks.

The neurons and the synapses can be considered the hardware of a neural network, while the weights and thresholds can be considered the software. To program a neural network, we choose a set of weights that makes the normal operation of the network simulate the computation we have in mind. If the choice of the weights and thresholds, in terms of the desired computation, could be automated, it would constitute a learning mechanism. For example, we could start with a set of training samples from the function we want to implement, and the learning mechanism would then choose the proper weights and thresholds that make the network simulate the function. This method would eliminate the need to design a new network each time we have a function to implement.

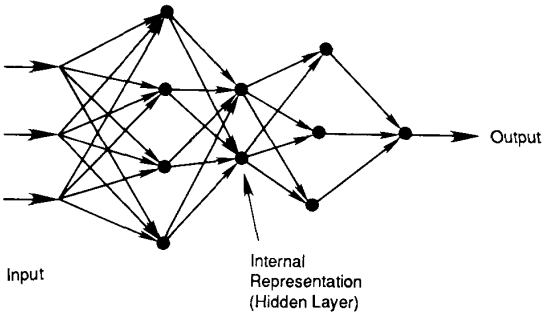

Fig. 2. Feed-forward network.

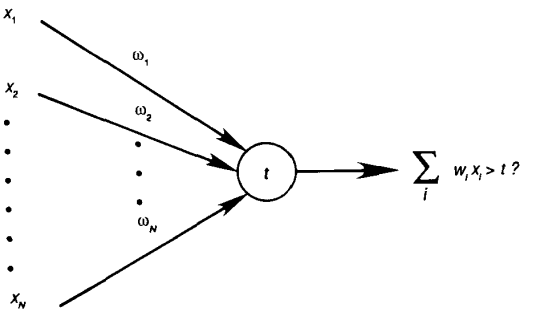

Fig. 3. Neuron.

While learning in general stands by itself as a research discipline, learning in feed-forward networks has been given special attention. Problems for which learning has the most potential to offer a solution are the natural variety, such as pattern recognition. For these problems, the representation of the data is crucial to the complexity of the problem. A picture can be represented just as a matrix of pixels, or it can be represented using a higher-level set of primitives that are better suited for recognizing the contents of the picture. In a feed-forward network, there are several internal representations of the data at each layer of pixels (Figure 2). This gradual transformation from raw data to higher levels of representation is very interesting, especially if the representations arise spontaneously via the learning mechanism.

While learning in general stands by itself as a research discipline, learning in feed-forward networks has been given special attention.

Memory Capacity

In contrast with the standard memory, where the amount of information storage is an explicit quantity, the information capacity of neural network models is a debatable concept. If we have a Random Access Memory (RAM) with M address lines and 1 data line (2^M memory locations, each storing 1 bit of information), the capacity is clearly 2^M bits. How do we define the capacity for a neural network?

The key is to look at the RAM in a different way. The RAM, as a whole, stores a string of 2^M bits. Therefore, it can distinguish between 2^{2^M} cases (the different ways of setting 2^M bits independently to 0 or 1). One can look at the capacity of the RAM as the logarithm of the number of cases it can distinguish between, namely $\log_2 2^{2^M} = 2^M$ bits. This definition treats the entire contents of the RAM as one object that encodes a message. The logarithm of the number of different messages is formally the information content of the message.

When we look at a feedback network in this way, the definition of capacity becomes apparent. While we read off the information by observing the state transitions, the information is contained in the choice of the weights and thresholds. It is that choice that determines which states go to which states. How many different sets of weights and thresholds can we distinguish between by observing the state transitions of the network? Using an argument for enumerating threshold functions, it can be shown following [1] that there are $2^{\alpha N^3}$

distinguishable networks of N neurons, where α is asymptotically a constant. Taking the logarithm, this means that the capacity of feedback network is proportional to N^3 bits.

This definition of capacity is a bit theoretical. In order to take advantage of this capacity, we need to encode the information in the state transitions. To read off the information, we observe which states go to which states and decode the message (Figure 4). This format of information storage is not practical, since the meaning is not apparent.

Another format for information storage in feedback networks, which is more practical, is stable states. As mentioned above, an energy function can be defined to show that state transitions lead to a stable state (a state where every neuron remains unchanged when it applies its update threshold rule). Stable states are vectors of bits that correspond to words in a regular memory. Indeed, convergence to stable states is the basis for using the feedback network as an associative memory. How many stable states can we store in a feedback network? We count only the states that we are free to choose, since these are the only ones that are relevant to information. The number of stable states is, essentially, βN (each consists of N bits, which are the individual states of the N neurons), where β is asymptotically a constant. Thus, in terms of bits, the stable-state capacity of a feedback network of N neurons is proportional to N^2 bits. It is not surprising that the capacity is down from N^3 to N^2, since we obviously lose information if we ignore the transitions that lead to a given stable state.

As we mentioned above, stable states that count are those that we can choose. There are algorithms that take a set of vectors and produce a network in which these vectors are stable states. One particularly simple one is the Hebbian rule, that amounts to choosing the matrix of weights to be the sum of outer products of the vectors to be stored. When we are forced to use the Hebbian rule, the stable-state capacity goes down further. The reason is that there are sets of vectors that can, in principle, be made stable, but for which the Hebbian rule fails. It can be shown that only $\gamma\ N/\log N$ randomly-chosen stable states can be stored using the Hebbian rule, where γ is asymptotically a constant [14]. In terms of bits, this capacity is proportional to $N^2/\log N$ bits.

For feed-forward networks, there is no definition of memory capacity that corresponds to stable-state capacity. After all, feed-forward networks do not have stable states, but rather input/output relations. For feed-forward networks with a single-bit output, the notion of discrimination capacity is introduced. Consider Figure 5, where a four-point function is represented geometrically. A threshold function corresponds to a line that separates the points that map to +1 (labeled X) from those that map to −1 (labeled O). Certain functions can be separated by a line (e.g., Figure 5a), while other functions cannot (e.g., Figure 5b). In higher dimensions, the separation is done using a hyperplane. The discrimination capacity of a neuron (or a simple threshold function) is measured by the maximum number of points that can be separated, in (almost) every possible way, using a hyperplane. this number is an indication of how big the set of input instances can be, while still expecting to simulate an arbitrary function on them using a single threshold. The discrimination capacity of a neuron turns out to be linear in the number of inputs [8]. Under simple assumptions, the discrimination capacity of a feed-forward network that has one hidden layer is approximately proportional to both the number of inputs and the number of neurons [4].

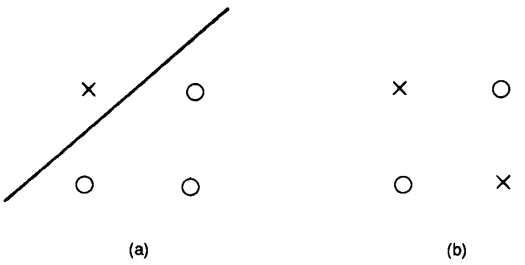

Fig. 5. Linearly separable and inseparable functions.

Computing Power

Most of the interest in neural networks arose from their use to perform useful computations. Roughly speaking, these computations fall into two categories; natural problems and optimization problems. Natural problems, such as pattern recognition, are typically implemented on a feed-forward network. The characterization of those functions that can be implemented on feed-forward networks is discussed in [13]. Two relevant parameters in the use of feed-forward networks for natural problems are; the discrimination capacity and the learning complexity, which are discussed elsewhere in this article.

Optimization problems are typically implemented on a feedback network. One famous example is the Traveling Salesman Problem (TSP), in which a salesman is supposed to tour a number of cities (visiting each exactly once, then returning to where he started) and desires to minimize the total distance of the tour (Figure 6). The intercity distances are given as the

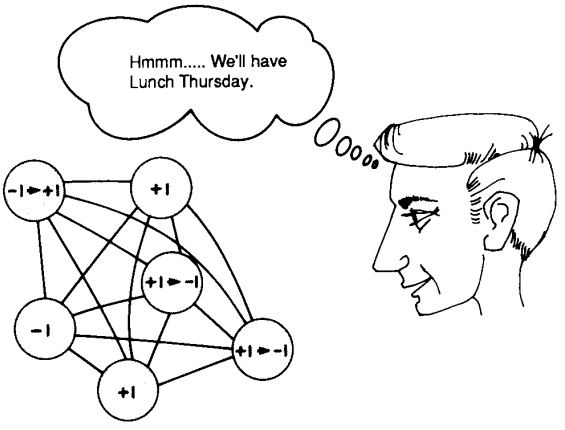

Fig. 4. Decoding a message.

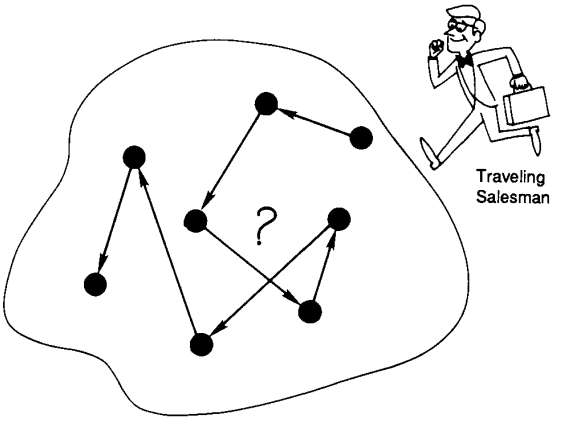

Fig. 6. Traveling salesman problem.

input, and the desired output is the shortest (or near-shortest) tour.

In order to implement a solution to the TSP, or any other optimization problem, on a feedback network, the energy function is used as a medium [10]. As we discussed above, the operation of the feedback network implies a descent on the energy surface. By designing the network so that the minimum of the energy function coincides with a minimum-length tour, the network becomes a computer that searches for the minimum tour. For small-size instances of the problem, there are reports of efficient neural network solutions to the TSP and other optimization problems. The solutions are vulnerable to the major problem of descent methods, namely, local minima.

It is possible to predict, theoretically, that feedback networks cannot offer good solutions to hard problems of large size [2] [7]. The reason is that if a fast neural-network solution exists, then one can show that a fast conventional-computer solution also exists, which it does not, by virtue of the assumption that the problem is hard. One interesting observation is that the often-appearing fast convergence of feedback networks to a stable state [12], which may sound like a computational advantage, is actually grounds to argue that neural networks cannot offer good solutions to hard problems. The point is that the complexity of the hard problem must be absorbed either in the network size or the convergence time, but, since the convergence time is small, the network size is forced to be prohibitively large.

Learning by Example

The idea of learning by example is illustrated in Figure 7. The goal is to produce a network (or, more generally, a system) that implements an unknown function, f, when given a sufficient number of input/output examples from the function. The process should be automated, as represented by a learning algorithm. The implementation may be only a good approximation of f.

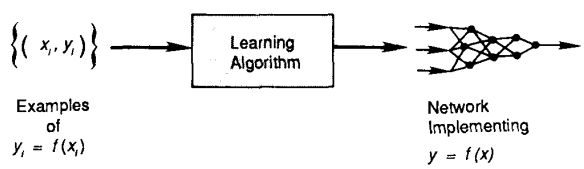

Fig. 7. Learning by example.

One of the popular algorithms for learning by example is back-propagation in feed-forward networks [15]. the algorithm operates on a network with a fixed architecture by changing the weights, in small amounts, each time an example $y_i = f(x_i)$ is received. The changes are made to make the response of the network to x_i closer to the desired output, y_i. This is done by gradient descent, and each iteration is simply an error signal propagating backwards in the network in a way similar to the input that propagates forward to the output. This fortunate property simplifies the computation significantly. However, the algorithm suffers from the typical problems of gradient descent, it is often slow and gets stuck in local minima.

There are two questions pertaining to learning by example; an information question and a complexity question. On the one hand, the learning algorithm is supposed to construct f from only partial information about f, namely, a number of examples $y_i = f(x_i)$. It is clear, however, that there are cases where the examples do not contain enough information about f. In this case, regardless of how clever the algorithm may be, it cannot produce an implementation of something it does not know. On the other hand, even if the examples contain enough information about f, the complexity of putting together this information to come up with an implementation of f may be prohibitive.

The information question can be posed in terms of generalization. Under what conditions will the performance of the network on the set of examples persist on previously unseen inputs? For example, suppose we are given few examples and we use a large network. From the discrimination capacity discussion, we expect to find a set of weights that implements any function on the few points (essentially, by memorizing the input/output examples). In this case, we should not really expect any generalization. The conditions under which generalization should be expected, have been studied using the notion of the Vapnik-Chervonenkis (VC) dimension [5] [6] [17]. Roughly speaking, for a fixed number of examples, the smaller the network, the better the generalization. This is not to say that a smaller network is more likely to implement the function, only more likely to behave similarly (for better or for worse) on a fresh input. The other conflicting requirement is that the network should be big enough to accommodate the function being implemented, regardless of the generalization question.

The complexity question can be posed in terms of polynomial time complexity. Under what conditions does there exist an algorithm that runs reasonably fast (e.g., in time that is polynomial in the size of the problem) to produce a network implementation of f from the sets of examples? The complexity of learning has been studied in [11] [16], and, except for extremely simple learning tasks, most of the results indicate that the learning complexity may be prohibitive. The experimental reports are mixed. Most of the learning tasks run sufficiently fast for small, yet interesting, problems. However, as the problem size increases, the computation time scales poorly. This observation is consistent with the theoretical predictions.

References

[1] Y. S. Abu-Mostafa and J. St. Jacques, "Information Capacity of the Hopfield Model," *IEEE Trans. Info. Theory*, vol. IT-31, pp. 461–464, 1985.

[2] Y. S. Abu-Mostafa, "Neural Networks for Computing?" *Neural Networks for Computing*, J. S. Denker, ed., New York: AIP Conf. Proc., vol. 151, pp. 1–6, 1986.

[3] Y. S. Abu-Mostafa and D. Psaltis, "Optical Neural Computers," *Scientific American*, vol. 256, no. 3, pp. 88–95, 1987.

[4] E. B. Baum, "On the Capabilites of Multilayer Perceptrons," *J. of Complexity*, Academic Press, vol. 4, pp. 193–215, 1988.

[5] E. B. Baum, and D. Haussler, "What Size Network Gives Valid Generalization," *Neural Computation*, MIT Press, vol. 1, pp. 151–160, 1989.

[6] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, "Classifying Learnable Geometric Concepts with the Vapnik-Chervonenkis Dimension," *Proc. ACM Symp. on Theory of Computing*, vol. 18, pp. 273–282, 1986.

[7] J. Bruck and J. Goodman, "On the Power of Neural Networks for Solving Hard Problems," *Neural Info. Processing Systems*, D. Z. Anderson, ed., New York: AIP, pp. 137–143, 1988.

[8] T. M. Cover, "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition," *IEEE Trans. Electron. Computers*, pp. 326–334, June 1965.

[9] J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Nat'l. Acad. Sci., USA*, vol. 79, pp. 2,554–2,558, 1982.

[10] J. J. Hopfield, "Collective Computation, Content-Addressable Memory, and Optimization Problems," *Complexity in Info. Theory*, Y. S. Abu-Mostafa, ed., Spring-Verlag, pp. 99–114, 1988.

[11] J. S. Judd, "On the Complexity of Loading Shallow Neural Networks," *J. of Complexity*, Academic Press, vol. 4, pp. 177–192, 1988.

[12] J. Komlos and R. Paturi, "Convergence Results in an Associative Memory Model," *Neural Networks*, vol. 1, no. 3, pp. 239–250, 1988.

[13] R. P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Mag.*, 1987.

[14] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh, "The Capacity of the Hopfield Associative Memory," *IEEE Trans. Info. Theory*, vol. 33, pp. 461–482, 1987.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing*, vol. 1, Cambridge, MA: MIT Press, 1986.

(Continued on page 82)

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, D. E. Rumelhart, J. L. McClelland, eds., Ch. 8, Cambridge, MA: MIT Press, 1986.
[8] J. Alspector, "Neural-Style Microsystems that Learn," this issue.
[9] H. Inose, *An Introduction to Digital Integrated Communications Systems*, University of Tokyo Press, pp. 114–117, 1979.
[10] M. Schwartz, *Telecommunication Networks: Protocols, Modeling, and Analysis*, pp. 526–552, Reading, MA: Addison-Wesley, 1987.
[11] E. Majani, R. Erlanson, and Y. Abu-Mostafa, "On the *K*-Winners-Take-All Network," *Advances in Neural Information Processing Systems I*, D. S. Touretzky, ed., pp. 634–642, San Mateo, CA: Morgan Kaufman, 1989.
[12] M. C. Paull, "Reswitching of Connection Networks," *Bell Syst. Tech. J.*, vol. 41, pp. 833–855, 1962.
[13] A. Marrakchi and T. Troudet, "A Neural Network Arbitrator for Large Crossbar Packet-Switches," *IEEE Trans. on Circuits and Syst.*, vol. 36, no. 7, pp. 1,039–1,041, July 1989.
[14] T. X. Brown, "Neural Networks Design Using The *K*-Winners-Take-All Circuit," submitted to the 1989 IEEE Neural Info. Processing Conf., Denver, CO.

Biography

Timothy X Brown received his B.S. degree in physics from Pennsylvania State University in 1986 and his M.S. degree in electrical engineering from CalTech in 1987. He is currently working on his Ph.D. at CalTech under the supervision of E. C. Posner. For his thesis, he is looking at new techniques for applying neural networks to problems in communications. Previous work includes a summer working for Bellcore in the Frequency Reuse Radio Systems Group where he studied system configurations for Universal Digital Portable Communications. His current research is supported by Pacific Bell.

(Continued from page 28)

[16] L. G. Valiant, "A Theory of the Learnable," *Commun. of the ACM*, vol. 27, pp. 1,134–1,142, 1984.
[17] V. N. Vapnik and A. Chervonenkis, "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities," *Theory of Prob. and Appl.*, vol. 16, pp. 264–280, 1971.

Biography

Yaser S. Abu-Mostafa is Associate Professor of Electrical Engineering and Computer Science at Caltech. In 1979 he received a B.Sc. degree in electrical engineering from Cairo University, in 1981 he received an M.S.E.E. from the Georgia Institute of Technology, and a Ph.D. in electrical engineering and computer science from Caltech in 1983, where he also received the Clauser Prize for the most original doctoral thesis. During the same year he joined Caltech as Assistant Professor and received the ASCIT teaching excellence award twice, in 1986 and 1989. Dr. Abu-Mostafa was the Program Chairman of the first annual IEEE Conference on Neural Information Processing Systems in 1987, and is currently Associate Editor for the *IEEE Transactions on Information Theory* and the *IEEE Transactions on Circuits and Systems*, he is also a member of the editorial boards of the *Journal of Complexity* and the *Neural Networks* journal, and a member of the IEEE Neural Networks Committee.