

Neural networks and learning

Yaser S. Abu-Mostafa

California Institute of Technology, Pasadena, California, USA 91125

ABSTRACT: The resurgence of interest in neural networks and learning over the last decade has had a profound impact on research work in computational devices and systems. Optical computers and analog VLSI owe their renewed vitality to the need to implement large-scale neural networks efficiently. It is the purpose of this paper to provide the basic background on neural networks and learning.

1. NEURAL NETWORKS

Neural network models offer an interesting alternative to performing certain computations. They have been particularly considered for unstructured computations [3] such as pattern recognition and artificial intelligence problems, and approximations to large optimization problems.

There are two popular models of neural networks, the feedback model [9] and the feedforward 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. The nodes are called neurons, and the edges are called synapses.

What characterizes a neural architecture in general, whether it is feedback or feedforward, is that the number of neurons is huge, and that each neuron performs a very simple task. In 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 feedforward model, the neurons are arranged into layers. 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 feedforward 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. The output y is determined by the inputs $x_1 \dots x_N$ according to a threshold rule. In the case of binary variables, the neuron simulates a function from $\{-1,+1\}^N$ to $\{-1,+1\}$ (we follow the neural network notation taking the binary convention to be +1s or -1s instead of 1s or 0s). The output depends on the input is through a set of real numbers called the weights $w_1 \dots w_N$, a weight for every input variable. If the sum $\sum_{i=1}^N w_i x_i$ exceeds an internal threshold t , 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 $\sum_{i=1}^N w_i x_i - t$ varies from large negative to large positive.

The operation of the feedforward 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 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.

The question of stability is crucial for feedback networks in order to guarantee a meaningful operation. The question does not arise in feedforward 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.

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 feedforward network. The characterization of which functions can be implemented on feedforward networks is discussed in [13]. 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. The intercity distances are given as the input, and the desired output is a 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.

2. LEARNING

While learning in general stands by itself as a research discipline, learning in feedforward 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 feedforward network, there are several internal representations of the data at each level of pixels. This gradual transformation from raw data to higher levels of representation is very interesting, especially if the representations arise spontaneously via the learning mechanism.

To define learning from examples in a formal way, we start by describing a simple setup. We have an *environment* such as the set of visual images, and we call the set X . In this environment we have a *concept* defined as a function $f : X \rightarrow \{0,1\}$, such as the presence or absence of a tree in the image. The goal of learning is to produce a *hypothesis*, also defined as a function $g : X \rightarrow \{0,1\}$, that approximates the concept f , such as a pattern recognition system that recognizes trees. To do this, we are given a number of examples $(x_1, f(x_1)), \dots, (x_N, f(x_N))$ from the concept, such as images with trees and images without trees.

In generating the examples, we assume that there is an unknown probability distribution P on the environment X . We pick each example independently according to this probability distribution. The statements in the theory hold true for any probability distribution P ,

which sounds very strong indeed. The catch is that the same P that generated the example is the one that is used to test the system, which is a plausible assumption. Thus we learn the tree concept by being exposed to 'typical' images.

The hypothesis g that we produce approximates f in the sense that g would rarely be significantly different from f . This definition allows for two tolerance parameters ϵ and δ . With probability $\geq 1 - \delta$, g will differ from f at most ϵ of the time. The δ parameter protects against the small, but nonzero, chance that the examples happen to be very atypical.

A learning algorithm is one that takes the examples and produces the hypothesis. The performance is measured by the number of examples needed to produce a good hypothesis as well as the running time of the algorithm. The running time of the learning algorithm is a key concern. As the number of examples increases, the running time generally increases. However, this dependency is a minor one. Even with few examples, an algorithm may need an excessive amount of time to manipulate the examples into a hypothesis. The independence of this complexity issue from the information issue is apparent. Without a sufficient number of examples, no algorithm slow or fast can produce a good hypothesis. Yet a sufficient number of examples is of little use if the computational task of digesting the examples into a hypothesis proves intractable.

REFERENCES

- [1] Y. S. Abu-Mostafa and J. St. Jacques, "Information capacity of the Hopfield model," *IEEE Trans. Inform. Theory*, vol. IT-31, pp. 461, 464, 1985.
- [2] Y. S. Abu-Mostafa, "Neural Networks for computing?" in *Neural Networks for Computing*, J. S. Denker (ed.), New York, AIP Conf. Proc., vol. 141, 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 capabilities of multilayer perceptrons," *Journal 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," *Prod. 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," in *Neural Information 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. Electronic Components*, pp. 326-334, June 1965.
- [9] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. USA*, vol. 79, pp. 2554-2558, 1982.
- [10] J. J. Hopfield, "Collective computation, content-addressable memory, and optimization problems," in *Complexity in Information Theory*, Y. S. Abu-Mostafa (ed.), Springer-Verlag, pp. 99-114, 1988.
- [11] J. S. Judd, "On the complexity of loading shallow neural networks," *Journal of Complexity*, Academic Press, vol. 4, pp. 177-192, 1988.
- [12] J. Kömlos and R. Paturi, "Convergence results in an associative memory model," *Neural Networks*, vol. 1, no. 3, pp. 239-250, 1988.
- [13] R. P. Lippman, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, 1987.
- [14] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh, "The capacity of the Hopfield associative memory," *IEEE Trans. Inform. Theory*, vol. 33, pp. 461-482, 1987.
- [15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, vol. 1, Cambridge, MA: MIT Press, 1986.
- [16] L. G. Valiant, "A theory of the learnable," *Communications of the ACM*, vol. 27, pp. 1134-1142, 1984.
- [17] V. N. Vapnik and A. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory of Probability and Its Applications*, vol. 16, pp. 264-280, 1971.