JOURNAL OF COMPLEXITY 6, 192–198 (1990)

Learning from Hints in Neural Networks*

YASER S. ABU-MOSTAFA

Departments of Electrical Engineering and Computer Science, California Institute of Technology, Pasadena, California 91125

Received April 3, 1989

Learning from examples is the process of taking input–output examples of an unknown function f and infering an implementation of f. Learning from hints allows for general information about f to be used instead of just input–output examples. We introduce a method for incorporating any invariance hint about f in any descent method for learning from examples. We also show that learning in a neural network remains NP-complete with a certain, biologically plausible, hint about the network. We discuss the information value and the complexity value of hints. © 1990 Academic Press, Inc.

1. INTRODUCTION

We can think of learning from examples as one end of a spectrum whose other end is explicit programming. Between these two extremes, there is a spectrum of largely unexplored possibilities.

To explain what we mean, let us assume that we have a decision function $f: X \rightarrow \{0, 1\}$ that we wish to implement; for instance, the primality problem where $X = \{1, 2, 3, \ldots\}$ and $f(x) = 1$ iff x is a prime, or the problem of recognizing a tree in an image where X is a set of images and $f(x) = 1$ iff x contains a tree. The goal is to come up with an implementation of f. We can write a simple program for f in the primality problem. However, the lack of a mathematical understanding of f in the image recognition problem forces us to seek other approaches. One approach is learning from examples, where we use a 'learning process' that we present with examples of images with trees and images without trees until

* Presented in part at the Third Symposium on Complexity of Approximately Solved Problems, April 3, 1989. Research supported by the Air Force Office of Scientific Research under Grant AFOSR-88-0213.

the process infers an implementation of f. Whenever we have an effective process for learning from examples, it is tantamount to automated programming. The process is a mechanical means of producing an implementation of f.

When feasible, learning from examples is a very convenient approach. It does not require any knowledge of f, just input–output examples. In many practical situations, we do have some knowledge of f. In these cases, it would be inefficient to take blind examples without taking advantage of what we already know about f. This gives rise to learning from hints as opposed to learning from examples. Learning from hints is still a *learning* process, since we do not know enough about f to program it outright.

A hint is any piece of information about f. As a matter of fact, an input–output example is a special case of a hint. A hint may take the form of a global constraint on f, such as a symmetry property or an invariance. It may also be partial information about the implementation of f.

A hint may be valuable to the learning process in two ways. It may reduce the number of functions that are candidates to be f (information value), and may reduce the number of steps needed to find the implementation of f (complexity value). For illustration, suppose we are learning about an unknown integer N ($10^5 < N < 10^6$) and we want to represent it as a six-digit number. Which is a more valuable hint: *The number is a prime* or *The most significant digit is 7?* Although the first hint has more information value, it may have less complexity value because it does not reduce the search space in a way that is easily compatible with the desired representation.

We shall report a positive result and a negative result in this paper. The positive result is a technique that incorporates any invariance hint in any descent technique for learning. The negative result is that general learning in neural networks remains NP-complete even with a hint that is biologically plausible. This paper provides a preliminary treatment of the subject of hints; many directions of investigation warrant further exploration.

For completeness, we briefly introduce feedforward neural networks and their gradient descent learning. A single-output feedforward neural network (Fig. 1) is a combinatorial circuit organized in layers of units (neurons). Each neuron performs a threshold function $\theta(\Sigma_j w_j u_j - t)$, where $\{u_j\}$ are the inputs to the neuron, $\{w_j\}$ are real numbers (weights), t is a real number (threshold), and θ is a 'sigmoid' function (soft or hard) that varies between -1 and $+1$ monotonically (the binary convention is ± 1 instead of 0, 1).

For any set of values for the weights and the thresholds, the network output y is a fixed function of the input variables $\mathbf{x} = x_1, \ldots, x_N$. In order to make the network implement a function $f(\mathbf{x})$, we need to choose the

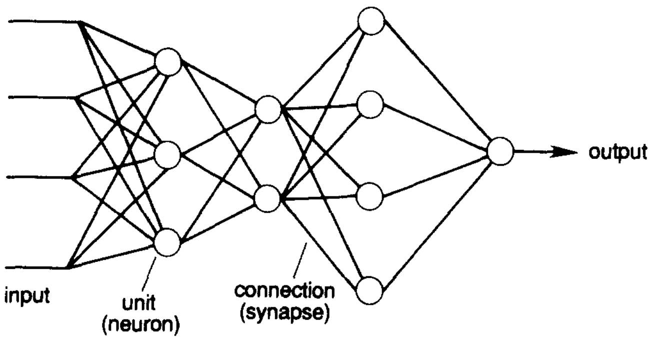

FIG. 1. Feedforward neural network.

weights and the thresholds such that the actual output y and the desired output $f(\mathbf{x})$ are as close as possible. A gradient descent method for learning f from examples minimizes $(y - f(\mathbf{x}))^2$ for each example by perturbing the weights and thresholds. The formula for perturbing the weights is

$$\Delta w_i \propto - \frac{\partial}{\partial w_i} (y - f(\mathbf{x}))^2 = -2(y - f(\mathbf{x})) \frac{\partial y}{\partial w_i} .$$

When this formula is applied to the neurons of a feedforward network, the resulting rule is the backpropagation algorithm of Werbos, described in (Rumelhart, Hinton, and Williams, 1986).

2. Invariance Hints

Many of the hints we have in pattern recognition problems are invariance hints. A hand written letter S can be deformed in many ways without losing its identity. Properties such as shift invariance, scale invariance, and rotation invariance are commonplace in image recognition.

An invariance property can be formalized as a set $\mathcal{A}$ of subsets $A \subset X$ such that if $x_1, x_2 \in A$, then $f(x_1) = f(x_2)$. Thus f is invariant within each set $A \in \mathcal{A}$. Interesting invariances are usually common to many functions f on the same domain X. For example, if X is the set of images, many recognition functions share some form of scale invariance. In this case, each set $A \in \mathcal{A}$ consists of images which are scaled versions of one another.

How can the invariance hint help the learning process? One way is to incorporate the hint directly in the implementation. For illustration, consider the implementation of a function f: $\{-1, +1\}^N \rightarrow \{-1, +1\}$ on a

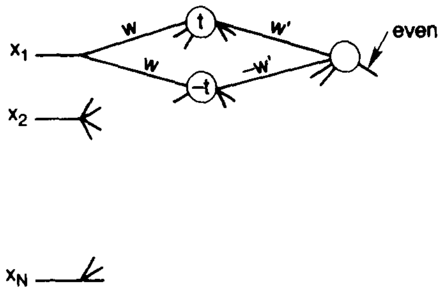

FIG. 2. Network for even functions.

feedforward neural network. If we know that f is an even function, i.e., $f(x_1, x_2, \ldots, x_N) = f(-x_1, -x_2, \ldots, -x_N)$, we can incorporate this hint in the network as shown in Fig. 2.

Each input x_i is applied to two sets of neurons, with one set implementing the dual functions of the other set. The dual of $f(x_1, x_2, \ldots, x_N)$ is defined as $-f(-x_1, -x_2, \ldots, -x_N)$, and can be implemented by using the same weights and the negative of threshold used to implement f. The outputs of the two dual neurons are then combined into the neurons of the next layer using a weight and its negative. From then on, the function implemented by the network is forced to be even in the variables $x_1, \ldots, x_N$. These constraints on the neurons of the first two layers must be taken into consideration in the learning process when examples of f are given. For instance, if we apply gradient descent, we cannot treat all the weights as independent variables any more.

Being an even function is much simpler than the invariance hints we are likely to encounter in pattern recognition. It may not be as easy to come up with a structure of the network that automatically guarantees a complicated invariance, such as elastic deformation. We will develop a unified method for incorporating any invariance, not in the structure of the network, but rather in the gradient descent method itself.

The key idea is expressing the hint itself as a set of examples. Suppose we are dealing with a function f: $\{-1, +1\}^N \to \{-1, +1\}$, which is invariant under cyclic shift of the input bits, e.g., $f(\mathbf{x}) = f(\mathbf{x}_2)$, where

$$\mathbf{x}_1 = -1\ -1\ +1\ +1\ -1\ +1\ -1\ +1\ -1\ -1\ -1\ -1,$$

$$\mathbf{x}_2 = -1\ -1\ -1\ -1\ +1\ +1\ -1\ +1\ -1\ +1\ -1\ -1.$$

The condition $f(\mathbf{x}_1) = f(\mathbf{x}_2)$ is an example of the hint as much as $f(\mathbf{x}) = +1$ would be an example of the function. Just like $f(\mathbf{x}) = +1$ can be enforced as a minimization of $(y - 1)^2$, where y is the output of the network when the input is $\mathbf{x}$, $f(\mathbf{x}_1) = f(\mathbf{x}_2)$ can be enforced as a minimization of $(y_1 - y_2)^2$, where y_1 and y_2 are the outputs of the network when the inputs are $\mathbf{x}_1$ and $\mathbf{x}_2$, respectively. The gradient descent perturbation of the weights in this case would be

$$\Delta w_i \propto -\frac{\partial}{\partial w_i}(y_1 - y_2)^2 = -2(y_1 - y_2)\left(\frac{\partial y_1}{\partial w_i} - \frac{\partial y_2}{\partial w_i}\right),$$

which is similar to the perturbation due to two examples of f.

The same idea is valid for all invariances and all descent techniques. It makes it possible to incorporate in a regular algorithm for learning from examples what would otherwise be a hard-to-implement invariance. The initial layers of the network can be dedicated to learning the invariance (the learning counterpart of the approach of Fig. 2).

A look at the expression for Δw_i reveals that *we do not need the actual value of f* in order to compute the perturbation of the weights due to an example of the invariance hint. This makes it possible to generate an arbitrary number of (possibly artificial) examples of the hint for the learning process without having to compute f (or even without the need to know which f we are learning). Such resource may be valuable if we have a limited number of natural examples of f.

This observation can be formalized within the framework of Vapnik and Chervonenkis (1971). If the set of candidate functions is significantly reduced by the constraint that they must satisfy the invariance property, the number of examples of f needed for the learning process decreases accordingly (Abu-Mostafa, 1989).

3. Complexity Issues

The process of learning an unknown function f in a feedforward network can be considered a search in Euclidean space for a set of weights and thresholds that implements the function. Indeed, if the domain of functions and networks is not restricted, the learning problem is NP-complete (Judd, 1988). Other, apparently simplistic, learning problems have also been shown to be NP-complete (Valiant, 1984).

Does the incorporation of hints in the learning process reduce the time complexity of learning? It is plausible that a hierarchical decomposition of the search space that results from learning different hints independently will reduce the search time. However, we were unable to show a meaningful instance of a hint that rendered an NP-complete learning problem polynomial-time. We will report on an interesting hint that did not change the NP-completeness of the problem.

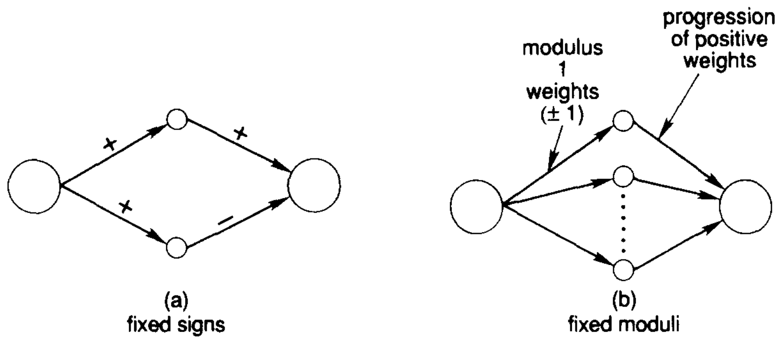

FIG. 3. Simulating arbitrary weights.

Consider the general problem of learning in a feedforward network with hard thresholds. Assume that, as part of the input to the learning process, we are given the *signs* of a set of weights that does implement the function in question. This hint is biologically motivated. In actual neurobiological systems, certain synapses are predisposed to be inhibitory and others to be excitory. Only the magnitude of the weight, not the sign, is left to the learning process.

The problem remains NP-complete as it turns out to be polynomially related to the old problem. To see this, we replace each synapse of the old network by the subnetwork of Fig. 3a. There are clearly choices of the weights with the prescribed signs that leads to an arbitrary equivalent weight for the original synapse.

It is interesting to note that the problem also remains NP-complete if we are given the absolute values of the weights and need only to find the signs! The reason is that each synapse can be replaced by the subnetwork of Fig. 3b. The prescribed moduli of the weights in this subnetwork can be given a pattern of signs that leads to an arbitrary equivalent weight (of finite accuracy) for the original synapse. Since we never need more than a polynomial number of bits for the weight (Hong, 1987), the polynomial equivalence is established.

4. CONCLUSION

Hints are pieces of information about an unknown function that we wish to learn, ranging from input–output examples to a complete implementation of the function. Invariance hints can be incorporated into descent methods of learning from examples, with possible gains in information and complexity. Certain strong hints do not change the NP-completeness status of learning.

Several directions of investigating the subject of hints remain open: to name a few, the compatibility of the hints with the desired implementation of f and with each other and how this affects their complexity value, the quantification of the information value of a hint in terms of the change in the V–C dimension, and finding natural examples of NP-complete learning problems that become polynomial-time using plausible hints.

Acknowledgment

I thank Dr. Robert Snapp for his assistance.

References

Abu-Mostafa, Y. S. (1988), Random problems, *J. Complexity* **4,** 277–284.

Abu-Mostafa, Y. S. (1989), The Vapnik–Chervonenkis dimension: Information versus complexity in learning, *Neural Comput.* **1.**

Hong, J. (1987), "On Connectionist Models," Technical Report (Computer Science), University of Chicago, Chicago, IL.

Judd, J. S. (1988), On the complexity of loading shallow neural networks. *J. Complexity* **4,** 177–192.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986), Learning internal representations by error propagation, *in* "Parallel Distributed Processing" (D. E. Rumelhart *et al.,* Eds.), Vol. 1," pp. 318–362, MIT Press, Cambridge, MA.

Valiant, L. G. (1984), A theory of the learnable, *Commun. ACM* **27,** 1134–1142.

Vapnik, V. N., and Chervonenkis, A. (1971), On the uniform convergence of relative frequencies of events to their probabilities, *Theory Probab. Appl.* **16,** 264–280.