JOURNAL OF COMPLEXITY **10**, 165–178 (1994)

Learning from Hints*

YASER S. ABU-MOSTAFA

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

Received October 1, 1992

DEDICATED TO JOSEPH F. TRAUB ON THE OCCASION OF HIS 60TH BIRTHDAY

We present a systematic method for incorporating prior knowledge (hints) into the learning-from-examples paradigm. The hints are represented in a canonical form that is compatible with descent techniques for learning. All the hints are fed to the learning process in the form of examples, and examples of the function are treated on equal footing with the rest of the hints. During learning, examples from different hints are selected for processing according to a fixed or adaptive schedule. Fixed schedules specify the relative emphasis of each hint, and adaptive schedules are based on how well each hint has been learned so far. We discuss adaptive minimization which is based on estimates of the overall learning error.

1. INTRODUCTION

It is evident that learning from examples needs all the help it can get. When an unknown function f is represented to us merely by a set of examples, we are faced with a dilemma. We want to use a model that is sophisticated enough to have a chance of simulating the unknown function, yet simple enough that a limited set of examples will suffice to "tune" it properly. These two goals are often on a collision course.

One established method of tackling this problem is regularization (Akaike, 1969). It is an attempt to start out with a sophisticated model and then restrict it during the learning process to fit the limited number of examples we have. Thus we have a simple model in disguise, and we

* This research was supported by the AFOSR under Grant F49620-92-J-0398.

0885-064X/94 $6.00

make it as simple as warranted by the resources. The hope is that the restriction (simplification) of the model has not rendered f impossible to simulate, and the justification is that this is the best we can do anyway given a limited set of examples.

Another method for tackling the problem is the use of hints (Abu-Mostafa, 1990, 1993) as a learning aid. Hints describe the situation where, in addition to the set of examples of f, we have prior knowledge of certain facts about f. We use this side information to our advantage. However, hints come in different shapes, and the main difficulty of using them is the lack of a systematic way of incorporating heterogeneous pieces of information into a manageable learning process. If what we know about f is that it is scale-invariant, monotonic over part of its domain, and represented by a given set of examples, we still have to integrate this information before we can learn the function. This paper concerns itself with the development of a systematic method that integrates different types of hints in the same learning process.

The distinction between regularization and the use of hints is worth noting. Regularization restricts the model in a way that is not based on known facts about f. Therefore, f may be implementable by the original model, but not by the restricted model. In the case of hints, if the unrestricted model was able to implement f, so would the model restricted by the hints, since f cannot be excluded by a litmus test that it is known to satisfy. We can apply any number of hints to restrict the model without risking the exclusion of f. The use of hints does not preclude, nor does it require, the use of any form of regularization.

How to take advantage of a given hint can be an art just like how to choose a model. In the case of invariance hints for instance, preprocessing of the input can achieve the invariance through normalization, or the model itself can be explicitly structured to satisfy the invariance (Minsky and Papert, 1969). Whenever such a method of direct implementation is feasible, the full benefit of the hint is automatically realized. This paper does not attempt to offer a superior alternative to direct implementation. *However, when direct implementation is not an option, we prescribe a systematic method for incorporating practically any hint in any descent method for learning.* The goal is to automate the use of hints in learning to a degree where we can effectively use a large number of simple hints that may be available in a practical situation.

We start by introducing the basic nomenclature and notation. The *environment* X is the set on which the unknown function f is defined. The points in the environment are distributed according to some probability distribution P. f takes on values from some set Y,

$$f: X \rightarrow Y.$$

Often, Y is just $\{0, 1\}$ or the interval $[0, 1]$. The *learning process* takes pieces of information about (the otherwise unknown) f as input and produces a *hypothesis* g

$$g: X \rightarrow Y$$

that attempts to approximate f. The degree to which a hypothesis g is considered an approximation of f is measured by a distance or "error,"

$$E(g, f).$$

The error E is based on the disagreement between g and f as seen through the eyes of the probability distribution P.

Two popular forms of the error measure are

$$E = \Pr[g(x) \neq f(x)]$$

and

$$E = \mathscr{E}[(g(x) - f(x))^2],$$

where $\Pr[.]$ denotes the probability of an event, and $\mathscr{E}[.]$ denotes the expected value of a random variable. The underlying probability distribution is P. E is always a non-negative quantity, and we take $E(g, f) = 0$ to mean that g and f are identical for all intents and purposes. We also assume that when the set of hypotheses is parameterized by real-valued parameters (e.g., the weights in the case of a neural network), E is well-behaved as a function of the parameters (in order to allow for derivative-based descent methods). We make the same assumptions about the error measures that will be introduced in section 2 for the hints.

In this paper, the "pieces of information" about f that are input to the learning process will be more general than those in the *learning from examples* paradigm. In that paradigm, a number of points $x_1, \ldots, x_N$ are picked from X (usually independently according to the probability distribution P) and the values of f on these points are provided. Thus, the input to the learning process is the set of examples

$$(x_1, f(x_1)), \ldots, (x_N, f(x_N))$$

and these examples are used to guide the search for a good hypothesis. In this paper, we consider the set of examples of f as only one of the available hints and denote it by H_0. The other hints $H_1, \ldots, H_M$ are additional known facts about f, such as invariance properties for instance.

The paper is organized as follows. Section 2 develops a canonical method for representing different hints. This is the first step in dealing with any hint that we encounter in a practical situation. Section 3 lays the foundations for learning from hints in general, and Section 4 presents specific implementations and experimental results. We discuss the overall picture in the conclusion.

2. Representation of Hints

We have so far described what a hint is in very general terms such as "a known property of f" or "a fact about f." Indeed, all that is needed to qualify as a hint for our purposes is to have a litmus test that f passes and that can be applied to the set of hypotheses. In other words, a hint H_m is formally a subset of the hypotheses, namely those satisfying the hint.

This definition of H_m can be extended to a definition of "approximation of H_m" in several ways. For instance, g can be considered to approximate H_m within ε if there is a function h that strictly satisfies H_m for which $E(g, h) \leq \varepsilon$. In the context of learning, it is essential to have a notion of approximation since exact learning is seldom achievable. Our definitions for approximating different hints will be part of the scheme for representing those hints.

The first step in representing H_m is to choose a way of generating "examples" of the hint. For illustration, suppose that H_m asserts that

$$f\colon [-1, +1] \rightarrow [-1, +1]$$

is an *odd* function. An example of H_m would have the form

$$f(-x) = -f(x)$$

for a particular $x \in [-1, +1]$. To generate N examples of this hint, we generate $x_1, \ldots, x_N$ and assert for each x_n that $f(-x_n) = -f(x_n)$. Suppose that we are in the middle of a learning process, and that the current hypothesis is g when the example $f(-x) = -f(x)$ is presented. We wish to measure how much g disagrees with this example. This leads to the second component of the representation, the error measure e_m. For the oddness hint, e_m can be defined as

$$e_m = (g(x) + g(-x))^2$$

so that $e_m = 0$ reflects total agreement with the example (i.e., $g(-x) = -g(x)$). The form of the examples of H_m and the choice of the error measure e_m are not unique.

Once the disagreement between g and an example of H_m has been quantified through e_m, the disagreement between g and H_m as a whole is automatically quantified through E_m, where

$$E_m = \mathscr{E}(e_m)$$

The expected value is taken w.r.t. the probability rule for picking the examples. This rule is also not unique. Therefore, E_m depends on our choices in all three components of the representation; the form of examples, the probability distribution for picking the examples, and the error measure e_m. Our choices are guided by certain properties that we want E_m to have. Since E_m is supposed to measure the disagreement between g and the hint, E_m should be zero when g is identical to f.

$$E = 0 \Rightarrow E_m = 0$$

This is a necessary condition for E_m to be consistent with the assertion that the hint is satisfied by f (recall that E is the error between g and f w.r.t. the original probability distribution P on the environment X).

Why is this condition necessary? Consider our example of the odd function f, and assume that the set of hypotheses contains even functions only. However, fortunately for us, the probability distribution P is uniform over $x \in [0, 1]$ and is zero over $x \in [-1, 0)$. This means that f can be perfectly approximated using an even hypothesis. Now, what would happen if we try to invoke the oddness hint? If we generate x according to P and attempt to minimize $E_m = \mathscr{E}[(g(x) + g(-x))^2]$, we will move towards the all-zero g (the only odd hypothesis), even if $E(g, f)$ is large for this hypothesis. This means that the hint, in spite of being valid, has taken us away from the good hypothesis. The problem of course is that, for the good hypothesis, E is zero while E_m is not. In other words, E_m does not satisfy the above consistency condition.

There are other properties that E_m should have. Suppose we pick a representation for the hint that results in E_m being identically zero for all hypotheses. This is clearly a poor representation in spite of the fact that it automatically satisfies the consistency condition! The problem with this representation is that it is extremely weak (every hypothesis "passes the $E_m = 0$ test" even if it completely disagrees with the hint). In general, E_m should not be zero for hypotheses that disagree (through the eyes of P) with H_m, otherwise the representation would be capturing a weaker version of the hint. On the other hand, we expect E_m to be zero for any g that does satisfy H_m, otherwise the representation would be stronger than the hint itself since we already have $E_m = 0$ when $g = f$.

On the practical side, there are other properties of the representation that are desirable. The probability rule for picking the examples should be

as closely related to P as possible. The examples should be picked independently in order to have a good estimate of E_m by averaging the values of e_m over the examples. Finally, the computation effort involved in the descent of e_m should not be excessive. In what follows, we illustrate these ideas by constructing representations for different types of hints.

Perhaps the most common type of hint is *the invariance hint*. This hint asserts that $f(x) = f(x')$ for certain pairs x, x'. For instance, "f is shift-invariant" is formalized by the pairs x, x' that are shifted versions of each other. To represent the invariance hint, an invariant pair (x, x') is picked as an example. The error associated with this example is

$$e_m = (g(x) - g(x'))^2.$$

A plausible probability rule for generating (x, x') is to pick x and x' according to the original probability distribution P conditioned on x, x' being an invariant pair.

There is another way to use the invariance hint that is particular to this type of hint. We can transform the examples of f itself in an invariant way, thus generating new examples of f. How well do these examples capture the hint? The transformed examples represent a restricted version of the invariance (restricted to the subset of the environment defined by the examples of f). On the other hand, the transformation takes advantage of the probability distribution that was used to generate the examples of f, which is usually the target distribution P. Using the invariance in this way and using it as an independent hint (represented to the learning process by its own examples) are not mutually exclusive.

Another related type of hint is the *monotonicity hint* (or inequality hint). The hint asserts for certain pairs x, x' that $f(x) \leq f(x')$. For instance, "f is monotonically nondecreasing in x" is formalized by all pairs x, x' such that $x \leq x'$. To represent a monotonicity hint, an example (x, x') is picked, and the error associated with this example is

$$e_m = \begin{cases} (g(x) - g(x'))^2 & \text{if } g(x) > g(x') \\ 0 & \text{if } g(x) \leq g(x') \end{cases}$$

It is worth noting that the set of examples of f can be formally treated as a hint, too. Given $(x_1, f(x_1)), \ldots, (x_N, f(x_N))$, the *examples hint* asserts that these are the correct values of f at these particular points x_n. Now, to generate an "example" of this hint, we independently pick a number n from 1 to N and use the corresponding $(x_n, f(x_n))$ (Fig. 1). The error associated with this example is e_0 (we fix the convention that $m = 0$ for the examples hint),

$$e_0 = (g(x_n) - f(x_n))^2.$$

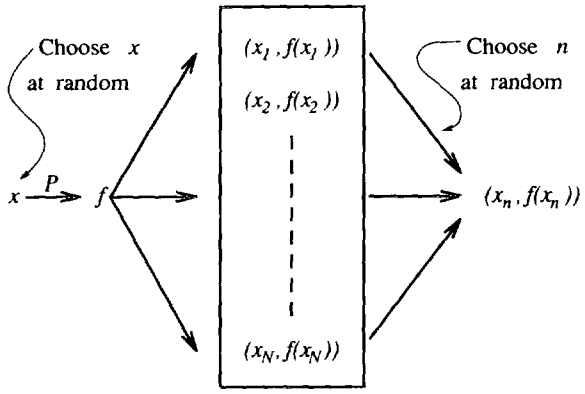

FIG. 1. Examples of the function as a hint.

Assuming that the probability rule for picking n is uniform over $\{1, \ldots, N\}$,

$$E_0 = \mathcal{E}(e_0) = \frac{1}{N} \sum_{n=1}^{N} (g(x_n) - f(x_n))^2$$

In this case, E_0 is also the best estimator of $E = \mathcal{E}[(g(x) - f(x))^2]$ given $x_1, \ldots, x_N$ that are independently picked according to the original probability distribution P. This way of looking at the examples of f justifies their treatment exactly as one of the hints, and underlines the distinction between E and E_0.

Another type of hint is the *approximation hint*. The hint asserts for certain points $x \in X$ that $f(x) \in [a_x, b_x]$. In other words, the value of f at x is known only approximately. The error associated with an example x of the approximation hint is

$$e_m = \begin{cases} (g(x) - a_x)^2 & \text{if } g(x) < a_x \\ (g(x) - b_x)^2 & \text{if } g(x) > b_x \\ 0 & \text{if } g(x) \in [a_x, b_x]. \end{cases}$$

The final type of hints that we discuss here arises when the learning model allows non-binary values for g where f itself is known to be binary. This gives rise to the *binary hint* (or Boolean hint). Let $\hat{X} \subseteq X$ be the set where f is known to be binary (for Boolean functions, $\hat{X}$ is the set of binary input vectors). The binary hint is represented by examples of the form x, where $x \in \hat{X}$. The error function associated with an example x

(assuming 0/1 binary convention, and assuming $g(x) \in [0, 1]$) is

$$e_m = g(x)(1 - g(x))$$

This choice of e_m forces it to be zero when $g(x)$ is either 0 or 1, while it would be positive if $g(x)$ is between 0 and 1. A natural probability rule for generating the examples is to pick x according to the original probability distribution P conditioned on $x \in \hat{X}$.

In a practical situation, we try to infer as many hints about f as the situation allows. Next, we represent each hint according to the guidelines discussed in this section. This leads to a list $H_0, H_1, \ldots, H_M$ of hints that are ready to produce examples upon the request of the learning algorithm. We now address how the algorithm should pick and choose between these examples as it moves along.

3. Learning Schedules

If the learning algorithm had complete information about f, it would search for a hypothesis g for which $E(g, f) = 0$. However, f being unknown means that the point $E = 0$ cannot be directly identified. The most any learning algorithm can do given the hints $H_0, H_1, \ldots, H_M$ is to reach a hypothesis g for which all the error measures $E_0, E_1, \ldots, E_M$ are zeros. Indeed, because of the consistency condition, $E = 0$ implies that $E_m = 0$ for all m.

If that point is reached, regardless of how it is reached, the job is done. However, it is seldom the case that we can reach the zero-error point because either (1) it does not exist (i.e., no hypothesis can satisfy all the hints simultaneously, which implies that no hypothesis can replicate f exactly), or (2) it is difficult to reach (i.e., the computing resources do not allow us to exhaustively search the space of hypotheses looking for this point). In either case, we have to settle for a point where the E_m's are "as small as possible."

How small should each E_m be? A balance has to be struck, otherwise some E_m's may become very small at the expense of the others. This situation would mean that some hints are over-learned while the others are under-learned. Knowing that we are really trying to minimize E, and that the E_m's are merely a vehicle to this end, *the criterion for balancing the E_m's should be based on how small E is likely to be as far as we can tell.*

It is important to distinguish between the quality of the hints and the quality of the learning algorithm that uses these hints. The quality of the hints is determined by how reliably we can predict that E will be close to

zero for a given hypothesis based merely on the fact that $E_0, E_1, \ldots, E_M$ are close to zero for that hypothesis. The quality of the algorithm is determined by how small $E_0, E_1, \ldots, E_M$ are likely to be for the hypothesis that will be produced by the algorithm within a reasonable time.

What any learning algorithm does in effect is to minimize the E_m's simultaneously. In order to discuss different algorithms in the right context, we first explore how simultaneous minimization of a number of quantities is done. Perhaps the most common method is that of *penalty functions* (Wismer and Chattergy, 1978). In order to minimize $E_0, E_1, \ldots, E_M$, we minimize the penalty function

$$\sum_{m=0}^{M} \alpha_m E_m,$$

where each α_m is a non-negative number that may be constant (*exact penalty function*) or variable (*sequential penalty function*). Any descent method can be employed to minimize the penalty function once the α_m's are selected. The α_m's are weights that reflect the relative emphasis or "importance" of the corresponding E_m's. The choice of the weights is usually crucial to the quality of the solution.

Even if the α_m's are determined, we still do not have the explicit values of the E_m's (recall that E_m is the expected value of the error e_m on an example of the hint). Instead, we *estimate* E_m by drawing several examples and averaging their error. Suppose that we draw N_m examples of H_m. The estimate for E_m would then be

$$\frac{1}{N_m} \sum_{n=1}^{N_m} e_m^{(n)},$$

where $e_m^{(n)}$ is the error on the nth example. Consider a batch of examples consisting of N_0 examples of H_0, N_1 examples of $H_1, \ldots$, and N_M examples of H_M. The total error of this batch is

$$\sum_{m=0}^{M} \sum_{n=1}^{N_m} e_m^{(n)}.$$

If we take $N_m \propto \alpha_m$, this total error will be a proportional estimate of the penalty function

$$\sum_{m=0}^{M} \alpha_m E_m.$$

In effect, we translated the weights into a *schedule*, where different hints are emphasized, not by magnifying their error, but by representing them with more examples.

We make a distinction between a *fixed* schedule, where the number of examples of each hint in the batch is predetermined (albeit time-invariant or time-varying, deterministic or stochastic), and an *adaptive* schedule where run-time determination of the number of examples is allowed (how many examples of which hint go into the next batch depends on how things have gone so far). For instance, constant α_m's correspond to a fixed schedule. Even if the α_m's are variable but predetermined, we still get a fixed (time-varying) schedule. When the α_m's are variable and adaptive, the resulting schedule is adaptive.

Finally, we can use *uniform batches* that consist of N examples of one hint at a time, or, more generally, *mixed batches* where examples of different hints are allowed within the same batch. If we are using a linear descent method with a small learning rate, a schedule that uses mixed batches is equivalent to a schedule that alternates between uniform batches (with frequency equal to the frequency of examples in the mixed batch). Figure 2 shows a fixed schedule that alternates between uniform batches giving the examples of the function (E_0) twice the emphasis of the other hints (E_1 and E_2). The schedule defines a turn for each hint to be learned. If we are using a nonlinear descent method, it is generally more

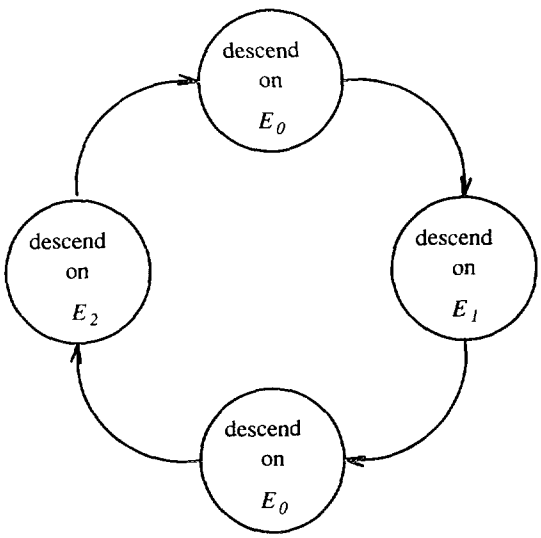

FIG. 2. A fixed schedule for learning.

difficult to ascertain a direct translation from mixed batches to uniform batches, but there may be compelling heuristic correspondences.

4. Implementations

In this section, we discuss different algorithms that use fixed and adaptive schedules to learn from hints. We introduce some easy-to-implement schedules. We also report experimental results about how hints affect learning.

The implementation of a given schedule (expressed in terms of uniform batches for simplicity) goes as follows: (1) The algorithm decides which hint (which m for $m = 0, 1, \ldots, M$) to work on next, according to some criterion; (2) the algorithm then requests a batch of examples of this hint; (3) it performs its descent on this batch; and (4) when it is done, it goes back to step 1. For fixed schedules, the criterion for selecting the hint can be 'evaluated' ahead of time, while for adaptive schedules, the criterion depends on what happens as the algorithm runs.

The following schedules were tested experimentally. We first describe the schedules and then report the results of the experiment.

Simple rotation. This is the simplest possible schedule that tries to balance between the hints. It is a fixed schedule that rotates between H_0, $H_1, \ldots, H_M$. Thus, at step k, a batch of N examples of H_m is processed, where $m = k \bmod (M + 1)$. This simple-minded algorithm tends to do well in situations where the E_m's are somewhat similar.

Weighted rotation. This is the next step in fixed schedules that tries to give different emphasis to different E_m's. The schedule rotates between the hints, visiting H_m with frequency ν_m. The choice of the ν_m's can achieve balance by emphasizing the hints that are more important or harder to learn. The schedule of Fig. 2 is a weighted rotation with $\nu_0 = 0.5$ and $\nu_1 = \nu_2 = 0.25$.

Maximum error. This is the simplest adaptive schedule that tries to achieve the same type of balance as simple rotation. At each step k, the algorithm processes the hint with the largest error E_m. The algorithm uses *estimates* of the E_m's to make its selection.

Maximum weighted error. This is the adaptive counterpart to weighted rotation. It selects the hint with the largest value of $\nu_m E_m$. The choice of the ν_m's can achieve balance by making up for disparities between the numerical values of the E_m's. Again, the algorithm uses estimates of the E_m's.

Adaptive schedules attempt to answer the question: Given a set of values for the E_m's, which hint is the most under-learned? The above schedules answer the question by comparing the individual E_m's. The adaptive minimization schedule answers the question by relating the E_m's to the actual error E.

Adaptive minimization. Given the estimates of $E_0, E_1, \ldots, E_M$, make $M + 1$ estimates of E, each based on all but one of the hints:

$$\begin{aligned}
&\hat{E}(\bullet, E_1, E_2, \ldots, E_M)\\
&\hat{E}(E_0, \bullet, E_2, \ldots, E_M)\\
&\hat{E}(E_0, E_1, \bullet, \ldots, E_M)\\
&\qquad\vdots\\
&\hat{E}(E_0, E_1, E_2, \ldots, \bullet)
\end{aligned}$$

and choose the hint for which the corresponding estimate is the *smallest*.

In other words, E becomes the common thread between the E_m's. If we use mixed batches, this translates to minimizing

$$\hat{E}(E_0, E_1, E_2, \ldots, E_M)$$

which has a gradient of

$$\frac{\partial \hat{E}}{\partial w} = \sum_{m=0}^{M} \frac{\partial \hat{E}}{\partial E_m} \frac{\partial E_m}{\partial w}.$$

Therefore, in gradient descent, the mixed batch would have examples from each hint in proportion to $\partial \hat{E}/\partial E_m$.

We ran a simple experiment to test the impact of hints on learning. We used basic gradient descent on a feed-forward neural network (back-propagation algorithm; Rumelhart *et al.*, 1986). The function f: $[-1, +1]^8 \rightarrow \{0, 1\}$ is defined by

$$f(x_1, \ldots, x_8) = \begin{cases} 1 & \text{if } -A < \sum_{n=1}^{8} x_n < A \\ 0 & \text{otherwise,} \end{cases}$$

where A is chosen to make $\Pr(f = 1) = \Pr(f = 0) = 0.5$. The input probability distribution P is uniform on $[-1, +1]^8$. The function f is both even and invariant under cyclic shift.

The network had 31 weights and thresholds (8–3–1 architecture). We ran the experiment for different training set sizes (20, 40, 80, 160, 320)

with no hints, evenness hint only, shift-invariance hint only, and both hints. In each of the 20 cases, we executed 100 runs and averaged the test error for all the runs in which the training set error went below 0.01, which happened 84% of the time. Each run consisted of 10,000 iterations, with each iteration being a uniform batch of 20 examples of either the function or one of the hints. We used the maximum-error schedule. The following table lists the test errors (in percentage).

	No hints	Evenness	Cyclic shift	Both hints
20 examples	46.3992	37.3563	27.5401	21.7994
40 examples	32.9017	16.0796	6.5705	2.3048
80 examples	10.1343	5.0666	2.1079	1.6349
160 examples	4.9589	2.8400	1.5461	1.3603
320 examples	2.4344	1.5073	1.2545	1.1082

To contrast the impact of hints to that of regularization, we ran the experiment again with no hints but with weight decay. We varied the parameter of the weight decay looking for the best performance. The resulting (percentage) test errors were 44.9690, 28.7600, 9.5683, 4.6430, and 2.6608, corresponding to 20, 40, 80, 160, and 320 examples, respectively.

5. Conclusion

The use of hints, under different names, is coming to the surface in a number of research communities dealing with learning and adaptive systems. The most common complaint about hints is that they are heterogeneous and cannot be easily integrated into learning. This paper was written with the specific goal of addressing this problem. The paper develops a systematic method for using different hints as input to the learning process. It treats all hints on equal footing, including the examples of the function. The most important features of the method are:

1. It makes no assumptions about what type of hints we can use. Rather, it gives a general procedure for representing the hint in a canonical way that is compatible with the common learning-from-examples paradigm.

2. It does not restrict the descent method we use. Rather, it provides a schedule for learning that is compatible with any descent method that may be suitable for the situation.

3. It is compatible with different models of learning, and with different techniques (such as regularization).

As the use of hints becomes routine, we are encouraged to exploit even the simplest observations that we may have about the function we are trying to learn. Since most hypotheses do not usually satisfy a given hint, the impact of hints is very strong in restricting the learning model in the right direction.

A number of algorithms for learning from hints were discussed in this paper. These algorithms use fixed or adaptive schedules to determine the turn of each hint to be learned. The goal of these schedules is to achieve balance between the errors of different hints. Adaptive schedules have the advantage of automatically compensating against many artifacts of the learning process. In particular, the adaptive minimization schedule is worth noting because it is based on estimating the actual test error.

ACKNOWLEDGMENT

I thank Ms. Zehra Cataltepe for her valuable input, especially in the experiments of Section 4.

REFERENCES

ABU-MOSTAFA, Y. S. (1990), Learning from hints in neural networks, *J. Complexity* **6,** 192–198.

ABU-MOSTAFA, Y. S. (1993), A method for learning from hints, *Adv. Neural Inform. Process. Systems* **5,** 73–80, Morgan-Kaufmann.

AKAIKE, H. (1969), Fitting autoregressive models for prediction, *Ann. Inst. Statist. Math.* **21,** 243–247.

MINSKY, M. L., AND PAPERT, S. A. (1969), "Perceptrons," MIT Press, Cambridge, MA.

RUMELHART, D. E., HINTON, G. E., AND WILLIAMS, R. J. (1986), Learning internal representations by error propagation, *in* "Parallel Distributed Processing," Vol. 1, pp. 318–362, MIT Press, Cambridge, MA.

SUDDARTH, S., AND HOLDEN, A. (1991), Symbolic neural systems and the use of hints for developing complex systems, *Internat. J. Machine Stud.* **35,** 291.

WISMER, D. A., AND CHATTERGY, R. (1978), "Introduction to Nonlinear Optimization," North-Holland, Amsterdam.