Monotonicity: Theory and Implementation

Joseph Sill
Yaser Abu-Mostafa

ABSTRACT 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. We focus in particular on the monotonicity hint, which states that the function to be learned is monotonic in some or all of the input variables. The application of monotonicity hints is demonstrated on two real-world problems- a credit card application task, and a problem in medical diagnosis. We report experimental results which show that using monotonicity hints leads to a statistically significant improvement in performance on both problems. Monotonicity is also analyzed from a theoretical perspective. We consider the class $\mathbf{M}$ of monotonically increasing binary output functions. Necessary and sufficient conditions for monotonic separability of a dichotomy are proven. The capacity of $\mathbf{M}$ is shown to depend heavily on the input distribution.

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 would like to use a model that is sophisticated enough to have a chance of simulating the unknown function, yet we want the model to be 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 [1]. 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 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 [2, 3] 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 wish to 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. This paper concerns itself with the development of a systematic method that integrates different types of hints in the same learning process. In particular, the monotonicity hint is presented from both practical and theoretical perspectives.

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 [4]. 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

$$f : X \to Y$$

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

$$g : X \to Y$$

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

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

and (for binary classification)

$$E = \mathcal{E}[-f(\mathbf{x})log(g(\mathbf{x})) - (1 - f(\mathbf{x}))log(1 - g(\mathbf{x}))]$$

where $\mathcal{E}[.]$ denotes the expected value of a random variable. The first measure is the well-known mean-squared error measure, while the second is a cross-entropy measure. The underlying probability distribution is P. E will always be a non-negative quantity, and we will take $E(g, f) = 0$ to mean that g and f are identical for all intents and purposes. We will 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 will be 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 $\mathbf{x_1}, \cdots, \mathbf{x_N}$ 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

$$(\mathbf{x_1}, f(\mathbf{x_1})), \cdots, (\mathbf{x_N}, \mathbf{f}(\mathbf{x_N}))$$

and these examples are used to guide the search for a good hypothesis. In this paper, we will consider the set of examples of f as only one of the available hints and denote it by H_0. The other hints $H_1, \cdots, H_M$ will be additional known facts about f, such as monotonicity 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. Section 4 presents specific implementations and experimental results for the case of the monotonicity hint. In section 5, we analyze how powerful a constraint monotonicity is from a theoretical viewpoint.

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 $E(g,h) \leq \epsilon$. 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

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

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 $x_1, \cdots, x_N$ 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 as well as 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

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

The expected value is taken w.r.t. the probability rule for picking the examples. This rule is also not unique. Therefore, E_m will depend 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 $E_m = \mathcal{E}[(g(x)+g(-x))^2]$, 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 will 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 $f(\mathbf{x}) = f(\mathbf{x}')$ for certain pairs $\mathbf{x}, \mathbf{x}'$. For instance, "f is shift-invariant" is formalized by the pairs $\mathbf{x}, \mathbf{x}'$ that are shifted versions of each other. To represent the invariance hint, an invariant pair $(\mathbf{x}, \mathbf{x}')$ is picked as an example. The error associated with this example is

$$e_m = (g(\mathbf{x}) - g(\mathbf{x}'))^2$$

A plausible probability rule for generating $(\mathbf{x}, \mathbf{x}')$ is to pick $\mathbf{x}$ and $\mathbf{x}'$ according to the original probability distribution P conditioned on $\mathbf{x}, \mathbf{x}'$ 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 $\mathbf{x}, \mathbf{x}'$ that $f(\mathbf{x}) \leq f(\mathbf{x}')$. For instance, "$f$ is monotonically nondecreasing in input variable x_i" is formalized by all pairs $\mathbf{x}, \mathbf{x}'$ such that $x_i \leq x_i'$. To represent a monotonicity hint, an example $(\mathbf{x}, \mathbf{x}')$ is picked, and the error associated with this example is

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

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

$$e_0 = (g(\mathbf{x_n}) - f(\mathbf{x_n}))^2$$

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

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

In this case, E_0 is also the best estimator of $E = \mathcal{E}[(g(\mathbf{x}) - f(\mathbf{x}))^2]$ given $\mathbf{x_1}, \cdots, \mathbf{x_N}$ 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.

In a practical situation, we try to infer as many hints about f as the situation will allow. Next, we represent each hint according to the guidelines discussed in this section.

3 MONOTONICITY HINTS

This section describes the application of a particular hint, the **monotonicity hint**, to two noisy real-world problems: a classification task concerned with credit card applications and a regression problem in medical diagnosis. Both databases were obtained via FTP from the machine learning database repository maintained by UC-Irvine [1].

[1] They may be obtained as follows: ftp ics.uci.edu. cd pub/machine-learning-databases. The credit data is in the subdirectory /credit-screening, while the liver data is in the subdirectory /liver-disorders.

The credit card task is to predict whether or not an applicant will default. For each of 690 applicant case histories, the database contains 15 features describing the applicant plus the class label indicating whether or not a default ultimately occurred. The meaning of the features is confidential for proprietary reasons. Only the 6 continuous features were used in the experiments reported here. 24 of the case histories had at least one feature missing. These examples were omitted, leaving 666 which were used in the experiments.

Intuition suggests that the classification should be monotonic in the features. Although the specific meanings of the continuous features are not known, we assume here that they represent various quantities such as salary, assets, debt, number of years at current job, etc. Common sense dictates that the higher the salary or the lower the debt, the less likely a default is, all else being equal. Monotonicity in all features was therefore asserted.

The motivation in the liver-diagnosis problem is to determine the extent to which various blood tests are sensitive to disorders related to excessive drinking. Specifically, the task is to predict the number of drinks a particular patient consumes per day given the results of 5 blood tests. 345 patient histories were collected, each consisting of the 5 test results and the daily number of drinks. The "number of drinks" variable was normalized to have variance 1. This normalization makes the results easier to interpret, since a trivial mean-squared-error performance of 1.0 may be obtained by simply predicting for mean number of drinks for each patient, irrespective of the blood tests.

The justification for monotonicity in this case is based on the idea that an abnormal result for each test is indicative of excessive drinking, where abnormal means either abnormally high or abnormally low.

In all experiments, batch-mode backpropagation with a simple adaptive learning rate scheme was used [2]. Several methods were tested. The performance of a linear perceptron was observed for benchmark purposes. For the experiments using nonlinear methods, a single hidden layer neural network with 6 hidden units and direct input-output connections was used on the credit data; 3 hidden units and direct input-output connections were used for the liver task. The most basic method tested was simply to train the network on all the training data and minimize the training error as much as possible. Another technique tried was to use a validation set to avoid overfitting. Training error for the credit problem was defined as

$$E_0 = \frac{1}{N}\sum_{n=1}^{N} -c_n log(g(\mathbf{x_n}) - (1-c_n) log(1-g(\mathbf{x_n}))$$

where c_n denotes the class label of example n. Mean squared error,

[2] If the previous iteration resulted in a decrease in error, the learning rate was increased by 3%. If the error increased, the learning rate was cut in half

$$E_0 = \frac{1}{N}\sum_{n=1}^{N}(g(\mathbf{x_n}) - t_n)^2$$

where t_n denotes the target output for example n, was used for the liver problem.

The linear perceptrons were trained for 300 batch-mode iterations each, while the nonlinear networks were trained for 5000 batch-mode iterations on the credit data and 1000 batch-mode iterations on the liver data. These amounts of training more than sufficed for the networks to approach regions near minima where the decrease in training error had become very slow.

A leave-k-out procedure was used in order to get statistically significant comparisons of the difference in performance. For each method, the data was randomly partitioned 200 different ways (The split was 550 training, 116 test for the credit data; 270 training and 75 test for the liver data). The results shown in Table 1 are averages over the 200 different partitions.

In the early stopping experiments, the training set was further subdivided into a set (450 for the credit data, 200 for the liver data) used for direct training and a second validation set (100 for the credit data, 70 for the liver data). The classification error on the validation set was monitored over the entire course of training, and the values of the network weights at the point of lowest validation error were chosen as the final values.

Training the networks with the monotonicity constraints was performed by adding a second term, E_1, to the objective function, so that the total objective function minimized was

$$E_0 + \lambda E_1$$

E_1 was a finite-sample approximation to $\mathcal{E}[e_1]$, the expected monotonicity error of the network. For each input variable, 500 pairs of vectors representing monotonicity in that variable were generated. The pairs asserting monotonicity were generated as follows: A single input vector $\mathbf{x}$ was generated from a multivariate gaussian model of the input distribution. The second vector, $\mathbf{x}$' was defined such that

$$\forall j \neq i, x'_j = x_j \tag{4}$$

$$x'_i = x_i + sgn(i)\delta x_i \tag{5}$$

where $sgn(i) = 1$ or -1 depending on whether f is monotonically increasing or decreasing in variable i, and δx_i was drawn from a $U[0,1]$ distribution.

For the lth pair,

$$e_1^l = \begin{cases} (g(\mathbf{x}) - g(\mathbf{x}'))^2 & \text{if } g(\mathbf{x}) > g(\mathbf{x}') \\ 0 & \text{if } g(\mathbf{x}) \leq g(\mathbf{x}') \end{cases}$$

This yielded a total of L=3000 hint example pairs for the credit problem and L=2500 pairs for the liver problem. λ was chosen, somewhat arbitrarily,to be 5000. No search for the optimal λ was attempted. E_1 was defined as the sum over all pairs of the monotonicity error on each pair:

$$E_1 = \frac{1}{L} \sum_{l=1}^{L} e_1^l$$

The process of training the networks with the monotonicity hints was divided into two stages. Since the meanings of the features were unaccessible, the directions of monotonicity were not known *a priori*. These directions were determined by training a linear perceptron on the training data for 300 iterations and observing the resulting weights. A positive weight was taken to imply increasing monotonicity, while a negative weight meant decreasing monotonicity.

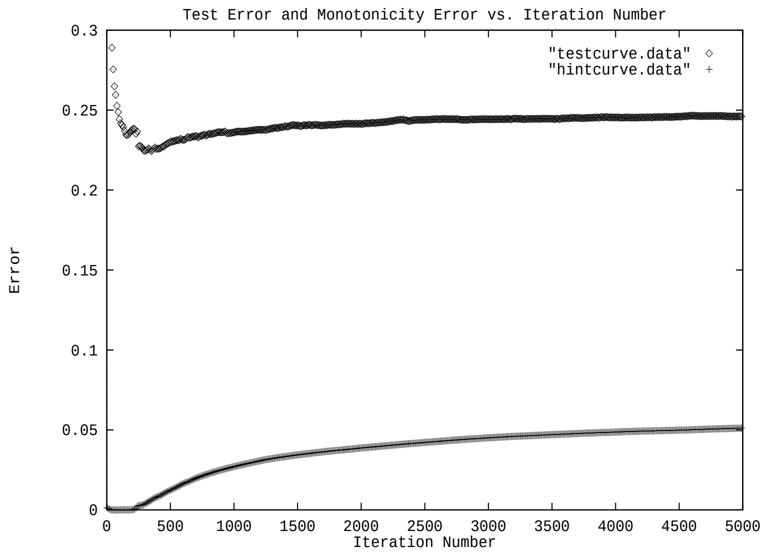

FIGURE 1. The violation of monotonicity tracks the overfitting occurring during training

Hint generalization, i.e. monotonicity test error, was measured by using 100 pairs of vectors for each variable which were not trained on but whose monotonicity error was calculated. For contrast, monotonicity test error was also monitored for the two-layer networks trained only on the input-output examples. Figure 1 shows test error and monotonicity error vs. training time for the credit data for the networks trained only on the training data, averaged over the 200 different data splits. The monotonicity error is multiplied by a factor of 10 in the figure to make it more easily visible. The figure indicates a substantial correlation between overfitting and monotonicity error during the course of training.

Method	training error	test error	hint test error
Linear	$22.7\% \pm 0.1\%$	$23.7\% \pm 0.2\%$	-
6-6-1 net	$15.2\% \pm 0.1\%$	$24.6\% \pm 0.3\%$	.005115
6-6-1 net, w/val.	$18.8\% \pm 0.2\%$	$23.4\% \pm 0.3\%$	-
6-6-1 net, w/hint	$18.7\% \pm 0.1\%$	$21.8\% \pm 0.2\%$	.000020

TABLE 1.1. Performance of methods on credit problem

Method	training error	test error	hint test error
Linear	$.802 \pm .005$	$.873 \pm .013$	-
5-3-1 net	$.640 \pm .003$	$.920 \pm .014$	.004967
5-3-1 net, w/val.	$.758 \pm .008$	$.871 \pm .013$	-
5-3-1 net, w/hint	$.758 \pm .003$	$.830 \pm .013$	.000002

TABLE 1.2. Performance of methods on liver problem

The performance of each method is shown in tables 1 and 2. Without early stopping, the two-layer network overfits and performs worse than a linear model. Even with early stopping, the performance of the linear model and the two-layer network are almost the same; the difference is not statistically significant. This similarity in performance is consistent with the thesis of a monotonic target function. A monotonic classifier may be thought of as a mildly nonlinear generalization of a linear classifier. The two-layer network does have the advantage of being able to implement some of this nonlinearity. However, this advantage is cancelled out (and in other cases could be outweighed) by the overfitting resulting from excessive and unnecessary degrees of freedom. When monotonicity hints are introduced, much of this unnecessary freedom is eliminated, although the network is still allowed to implement monotonic nonlinearities. Accordingly, a modest but clearly statistically significant improvement on the credit problem (nearly 2%) results from the introduction of monotonicity hints. Such an improvement could translate into a substantial increase in profit for a bank. Monotonicity hints also significantly improve test error on the liver problem; 4% more of the target variance is explained.

4 Theory

Much of learning theory is concerned with determing the number of examples needed for good generalization. Concepts such as capacity [5], VC-dimension [6], and effective number of parameters [7] have been developed in order to get bounds on the generalization (test error) given the training error and the model complexity. The ideas of capacity and VC-dimension also help to explain the benefit of introducing hints [8] into the learning process. By constraining a model to obey a hint, the model's capacity is reduced without an accompanying reduction in its ability to approximate the target function [9].

We present results here concerning the reduction in capacity associated with the use of the monotonicity hint. We showed in the preceding section that the use of monotonicity hints can yield significant improvement in the performance of neural networks on real-world tasks. To quantify how powerful a constraint monotonicity is would therefore be of some interest.

We will consider the class **M** of monotonically increasing (non-decreasing) functions from $R^d \to \{0, 1\}$. We will say that $\mathbf{x}' \geq \mathbf{x}$ if $\forall i, 1 \leq i \leq d, x'_i \geq x_i$. **M** is the class of all functions such that

$$\mathbf{x}' \geq \mathbf{x} \to f(\mathbf{x}') \geq f(\mathbf{x})$$

In many applications, prior information may indicate decreasing monotonicity in some variables rather than increasing monotonicity. The analysis which follows will also hold for each of the other $2^d - 1$ function classes where some or all variables have a decreasing monotonicity constraint rather than an increasing one. This equivalence is made clear by observing that decreasing monotonicity may be converted to increasing monotonicity by relabelling an input variable as its negation. There are also many situations where monotonicity only holds for some variables, while the relationship of the other variables to the output is completely unknown a priori. This case is more complex and will not be addressed here. Note that the class **M** is not explicitly parametrized by weights, unlike classes such as sigmoidal networks with a fixed number of hidden units. When a finite, parametrized model is further constrained to obey monotonicity in all variables, the resulting class of functions will be some subset of **M**. Thus, bounds on the capacity of **M** will upper-bound the capacity of any parametrized model where monotonicity hints are enforced.

Subsection II derives results about the capacity of **M**. In particular, the capacity is shown to depend almost completely on the input distribution. Subsection III describes the decision boundaries of monotonic classifiers, comparing and contrasting them to separating hyperplanes.

4.1 Capacity results

To make the idea of capacity precise, we must define a few auxiliary concepts. Define a *dichotomy* to be a set of d-dimensional input vectors, each of whom have an associated class label of either 0 or 1. Define a positive example as an input vector labelled 1 and a negative example as an input vector labelled 0. We say that a dichotomy is separable by a function class if there exists at least one function in the class which maps each of the input vectors to its correct class label. Define a random dichotomy as a set of input vectors drawn from the input distribution, with the label 0 or 1 assigned randomly with equal probability. Let $P(n)$ be the probability that a random dichotomy of n examples may be separated by the function class. We call the capacity the integer n^* for which $P(n^*)$ is closest to 0.5. Capacity (unlike VC-dimension) is therefore a quantity which *depends on the input distribution.* The importance of this point will become clear below.

The following theorem holds about monotonic separability:

Theorem 1 *A dichotomy is separable by* $\mathbf{M}$ *if and only if there exists no negative example which is greater than or equal to some positive example*

The necessity of this condition is obvious. Sufficiency may be demonstrated by constructing a function belonging to $\mathbf{M}$ which implements the dichotomy. Consider the function f which classifies as 1 only those input vectors which are greater than or equal to some positive example in the dichotomy. By construction, f obviously separates the dichotomy correctly. It is also clear that f belongs to $\mathbf{M}$. Consider an input vector $\mathbf{x}$ which f classifies positively. $\mathbf{x}$ must be greater than or equal to some vector $\mathbf{x}^*$ in the dichotomy. Any vector $\mathbf{x}'$ which is greater than or equal to $\mathbf{x}$ must also be greater than or equal to $\mathbf{x}^*$. Thus, $f(\mathbf{x}') \geq f(\mathbf{x})$, i.e., f belongs to $\mathbf{M}$.

Capacity results for the perceptron (i.e., the class of linear threshold functions) are well known: the capacity tends to $2d$, where d is the dimensionality of the input space [5]. This result is true for any smooth distribution of continuous input variables, because the number of linearly separable dichotomies of a set of input vectors is independent of how those input vectors are arranged, assuming no degenerate cases. This independence with respect to input distribution is in sharp contrast to the case of $\mathbf{M}$. Here, the number of dichotomies depends *heavily* on how the input vectors are arranged.

Consider figure 2. A little inspection will reveal that this dichotomy is not monotonically separable, even if we are free to take decreasing monotonicity to hold for one or both of the input variables. Figure 3 depicts a drastically different situation. In this case, there exists a monotonically *decreasing* relationship between the two input variables. When n input poin ts are arranged according to such a distribution, any of the 2^n possible dichotomies are separable by $\mathbf{M}$, the class of monotonically *increasing* functions. For a dichotomy not to be separable by $\mathbf{M}$, there must be a negative example which is greater than some positive example. But if the two input

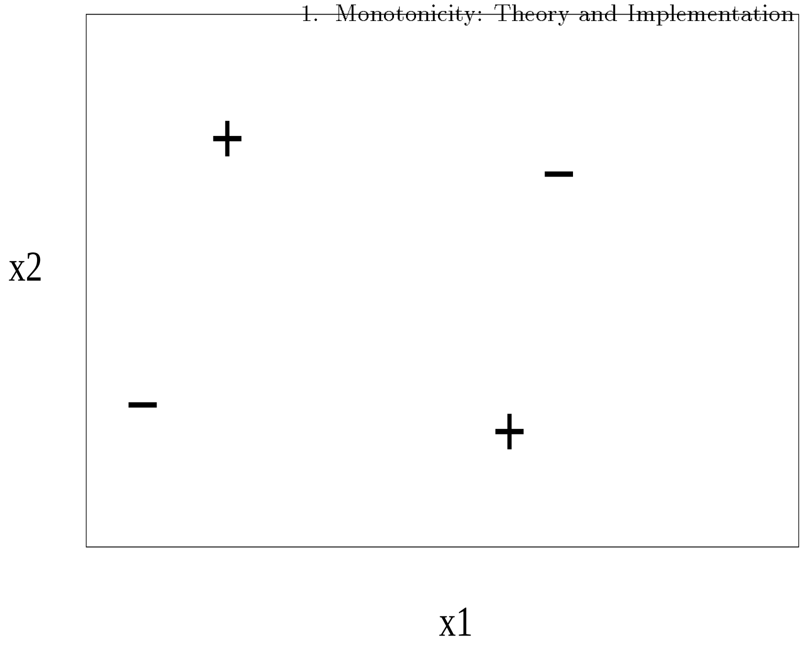

FIGURE 2. This dichotomy of four points cannot be implemented by any monotonic function, regardless of whether the monotonicity in each variable is increasing or decreasing. x's indicate positive examples, o's negative examples.

variables are related in a monotonically decreasing way, there are no two examples such that one is greater than the other in the vector sense, i.e., in both input variables. Therefore all dichotomies are separable given this input distribution. Note that even if the input dimensionality is greater than 2, a monotonically decreasing relationship between two of the input variables is sufficient to make all dichotomies separable by **M**. The above argument still holds, and monotonicity in the other $d-2$ variables may be satisfied by ignoring these variables, i.e., by choosing a function which is constant in these variables.

The example depicted in figure 3 establishes a result which explains why the input distribution-dependent notion of capacity is more useful than the concept of VC-dimension for the analysis of monotonicity. Recall that the VC-dimension is the maximum value of n for which the growth function $G(n) = 2^n$. The growth function is defined by the maximum number of separable dichotomies of n points, where this maximum is taken over all possible arrangements of the n points. Figure 3 demonstrates that we may always arrange input points in such a way that all 2^n possible dichotomies may be separated monotonically. It is granted that the figure 3 example is not very realistic- it would be extremely odd to find a problem where the target is believed to increase monotonically with two input variables which appear to be related to each other in a monotonically decreasing way. Nonetheless, such an example is permitted by the definition of the growth

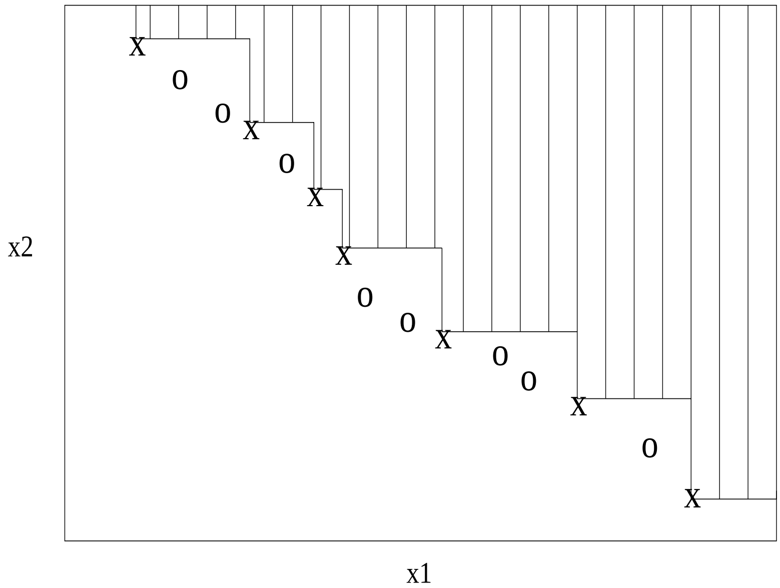

FIGURE 3. If two input variables are related in a monotonically decreasing way, then any dichotomy may be separated by M. A particular, arbitrary dichotomy is shown here. The shaded area indicates the region of input space which, by monotonicity, must be classified positively.

function. Thus, the VC-dimension of the class of monotonic functions = ∞ ! This result is misleading, however, since monotonicity is still a very powerful constraint in most cases.

Figure 4 depicts a situation opposite to that of figure 3. In this case, there exists a monotonically *increasing* relationship between the two input variables. **M** has very little separating power in such a situation. The second input variable adds nothing- the only dichotomies separable by **M** are those which were already separable using only one of the input variables. The class **M** is free only to choose a dividing point from among the n input vectors and classify positively those vectors greater than or equal to the dividing vector. Thus, only $n+1$ of the 2^n possible dichotomies is separable by **M**. By a similar argument, the number of dichotomies is separable by **M**. By a similar argument, the number of separable dichotomies remains $n+1$ if we have d input variables all of which are related to each other in monotonically increasing way, i.e., all of which rise and fall together.

The preceding examples demonstrate that the number of dichotomies separable by **M**, and hence, the capacity of **M** can be arbitrarily large or small depending on the particular input distribution. The second example and especially the third example cannot be dismissed as merely irrelevant, degenerate cases which will never occur in real life. These two examples are the extreme versions of possible real world situations where the input

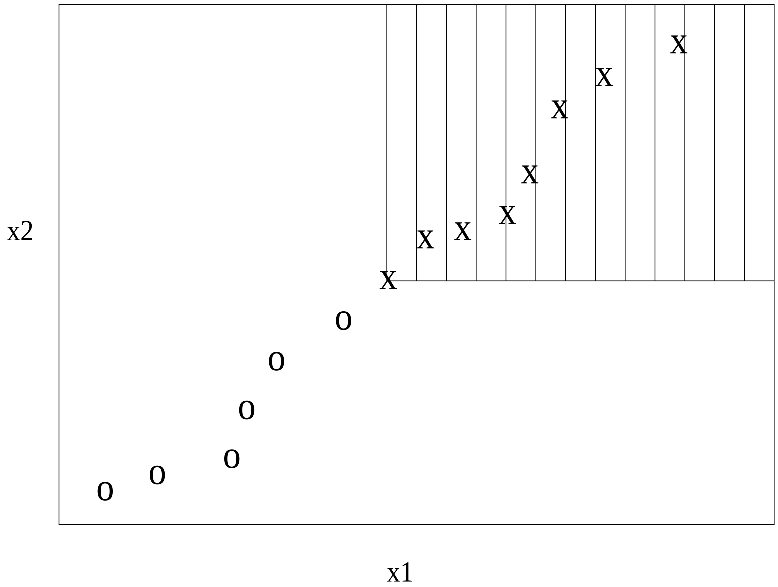

FIGURE 4. If two input variables are related in a monotonically increasing way, then only $n+1$ of the 2^n possible dichotomies are separable by $\mathbf{M}$. A particular dichotomy is pictured here. The shaded area indicates the region of input space, by monotonicity, must be classified positively.

variables do not have strict monotonic relationships but are correlated significantly. It should be clear from figure 4 that if we have two input variables which are highly but not perfectly positively correlated, then the number of separable dichotomies of n points is still likely to be low, although somewhat higher than $n+1$. Likewise, if we have two input variables which have a large, negative correlation, then the number of separable dichotomies is likely to be quite high, although somewhat less than 2^n.

The total dependence of the capacity of $\mathbf{M}$ on the input distribution makes it somewhat unlikely that much more useful analytic work on this topic can be done. If we have a good model for the input distribution for a given problem, however, the capacity of $\mathbf{M}$ may be estimated computationally. n input vectors may be drawn from the model of the input distribution and labelled randomly as positive or negative with equal probability. Theorem 1 tells us that we can then check whether or not the dichotomy generated is separable by $\mathbf{M}$. This procedure may be repeated many times in order to estimate $P(n)$, the probability that n randomly labelled points are separable by $\mathbf{M}$. The capacity of $\mathbf{M}$ is then that n^* for which $P(n^*) \approx 0.5$.

An estimate of $P(n)$ may also be used to make explicit statements about the generalization of a monotonic model. The VC bounds on generalization, based on the growth function, are well known. Tighter bounds may be obtained by replacing the growth function with the VC entropy, i.e., the expected number of dichotomies [10]. The expected number of dichotomies separable by M is simply $2^n P(n)$. Thus, the estimate of $P(n)$ could be used to get real bounds on generalization error given training error when a monotonic model is used.

4.2 Decision boundaries

A monotonic classifier may be thought of as a mildly nonlinear generalization of a linear classifier. This relationship is perhaps best demonstrated by considering the decision boundaries corresponding to the two models. It is well known that the decision boundary implemented by a linear perceptron is simply a $d-1$ dimensional hyperplane splitting input space into two regions. In two dimensions, this boundary consists of a straight line dividing the input plane.

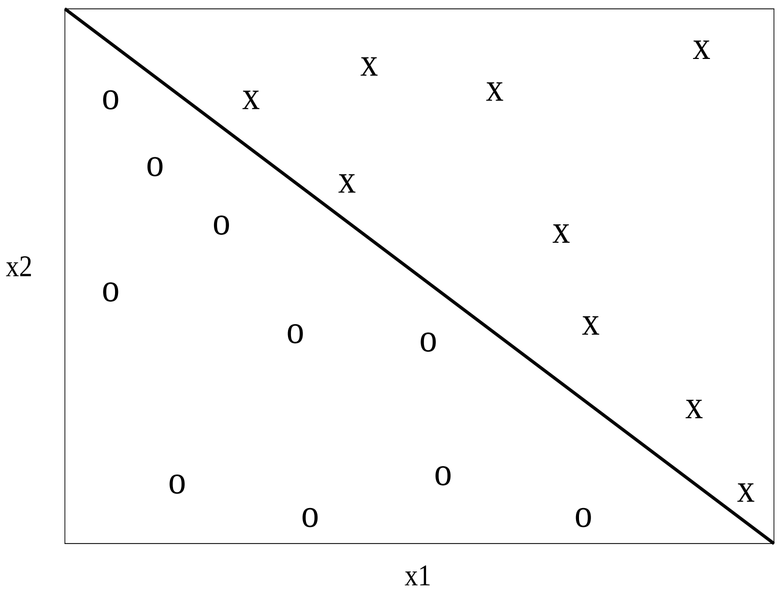

FIGURE 5. A linear classifier splits the input plane with a line

Consider a classifier $m(x_1, x_2)$ which is monotonically increasing in both input variables. We will also place one additional, mild constraint on m. Assume that for any x_1, there exists some x_2 such that $m(x_1, x_2) = 0$ and some x_2 such that $m(x_1, x_2) = 1$. Assume also for any x_2, there exists some x_1 such that $m(x_1, x_2) = 0$ and some x_1 such that $m(x_1, x_2) = 1$. If the above conditions hold, then the decision boundary implemented by $m(x_1, x_2)$ must be a one-to-one mapping between x_1 and x_2 [Fig 4]. Let x_1 be fixed, and let $x_2^{min}(x_1)$ be the smallest value x_2 can take such that $m = 1$. By monotonicity, for any $x_2 \geq x_2^{min}$, m also equals 1. Thus, x_2^{min}

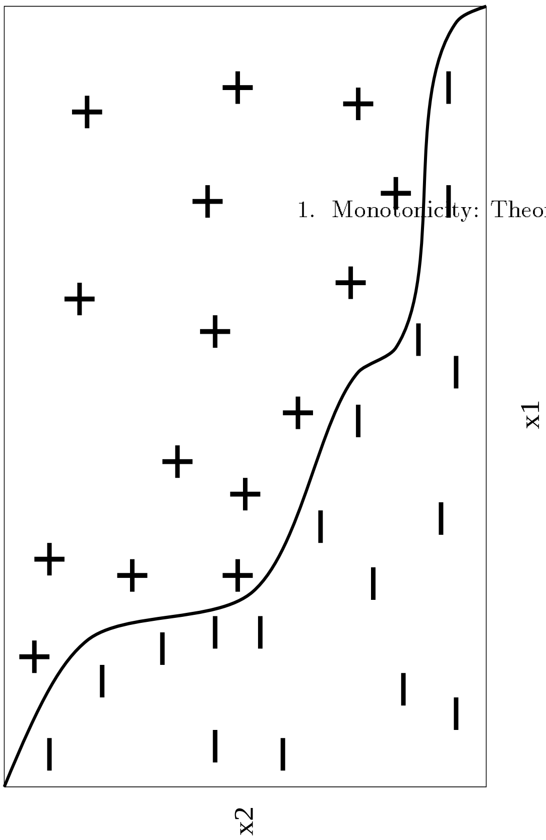

FIGURE 6. A monotonic classifier splits the input plane with a decision boundary consisting of a one-to-one mapping between the two input variables. Monotonic classifiers are analogous to but slightly more flexible than linear classifiers, which split the input plane with a straight line.

is the only value of x_2 for x_1 fixed where the classification changes. By the same argument, there exists a single $x_1^{min}(x_2)$ for every x_2. Therefore, the decision boundary must be an invertible function from one input variable to the other. This result makes clear our intuition that a monotonic model is more flexible than a linear one, but is still very severely constrained. The decision boundary of a monotonic classifier in the general case of d dimensions is somewhat harder to summarize, but some statements can be made. If we make analogous assumptions, then the boundary must be one continuous sheet such that specifying the values of any subset of $d-1$ input variables defines a unique value of the dth variable at which the classification changes. Thus, the boundary is a generalization to d dimensions of an invertible function.

5 CONCLUSION

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. The monotonicity hint provides an excellent illustration of the above point. We showed how the monotonicity constraint improved performance on real data. We also analyzed the power of the constraint from a theory persepective, demonstrating how a monotonic decision boundary is much more restricted than an arbitrary decision boundary which might be produced by a neural network with no prior knowledge supplied. Future directions may include the extension of this work to other hints, i.e., both theoretical and practical demonstrations of their benefits.

Acknowledgements

The authors thank Eric Bax, Zehra Cataltepe, Malik Magdon-Ismail, and Xubo Song for many useful comments.

6 References

[1] Akaike, H. Fitting autoregressive models for prediction, *Ann. Inst. Stat. Math.* 21:243-247,1969.

[2] Abu-Mostafa, Y. S. Learning from hints in neural networks. *Journal of Complexity* 6:192-198,1990.

[3] Abu-Mostafa, Y. S. A method for learning from hints In S. Hanson et al,editors, *Advances in Neural Information Processing Systems 5*, pp. 73-80 , 1993.

[4] Minsky, M. L. and Papert, S. A. *Perceptrons*, MIT Press , 1969.

[5] Cover, T.M. Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. *IEEE Transactions on Electronic Computers* 14:326-334,1965.

[6] Vapnik, V.N. and Chervonenkis, A.Y On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability and Its Applications* 16:264-280,1971.

[7] Moody, J.E. The *Effective* Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems. In J.E. Moody, S.J. Hanson, R.P. Lippman, editors, *Advances in Neural Information Processing* 4:847-854. Morgan Kaufmann, San Mateo, CA, 1992.

[8] Abu-Mostafa, Y.S. Hints *Neural Computation* 7:639-671,1995.

[9] Abu-Mostafa, Y.S. Hints and the VC Dimension *Neural Computation* 4:1993

[10] Vapnik, V. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, NY, 1995.