Monotonicity Hints for Credit Screening

Joseph Sill†, Yaser Abu-Mostafa‡
† Computation and Neural Systems program, California Institute of Technology
Pasadena, CA
‡ Departments of Electrical Engineering and Computer Science, California Institute of Technology
Pasadena, CA

Abstract— **A hint is any piece of side information about the target function to be learned. We describe an application of monotonicity hints to a real world problem. The task considered is the screening of credit card applicants. A measure of the monotonicity error of a candidate function is defined and an objective function for the enforcement of monotonicity is derived from Bayesian principles. We report experimental results which show that using monotonicity hints leads to a statistically significant improvement in performance on the credit screening problem.**

1 Introduction

Designing a machine learning model involves balancing two conflicting concerns [1]. The model used should be powerful enough to implement the details of the unknown target function f, but simple enough that the optimal parameters for doing so may be estimated accurately from the available data.

The use of hints [2] in the learning process allows us to satisfy the second requirement better without compromising the first. A hint is any piece of information known about f beyond the available input-output examples. For instance, f may be known to be invariant or symmetric with respect to some transformation of the input. The constraint imposed by a hint results in a model which is effectively simpler and has fewer degrees of freedom [3]. However, no loss in the ability to implement f is incurred[1] , since f is known to satisfy the constraint. Invariances in character recognition [4] and symmetries in financial market forecasting [5] are some of the instances of hints which have proved beneficial to learning systems applied to real-world problems.

Monotonicity is another hint which suggests itself in many applications. In screening credit card applicants, for instance, one might expect that the probability of default decreases monotonically with the applicant's salary. This paper describes the incorporation of monotonicity hints into a learning system applied to a noisy real-world classification task concerned with credit card applications. Artificial neural networks are the learning models used here, although the method may be used with other models as well.

Section II derives, from Bayesian principles, an appropriate objective function for simultaneously enforcing monotonicity and fitting the data. Section III discusses the rationale behind the assertion of monotonicity for the credit card application and describes the details and results of the experiments. Section IV analyzes the results and considers possible future directions.

2 Bayesian Interpretation of Objective Function

Let $\mathbf{x}$ be a vector drawn from the input distribution and $\mathbf{x}'$ be such that

$$\forall j \neq i, x'_j = x_j \tag{1}$$

$$x'_i > x_i \tag{2}$$

The statement that f is monotonically increasing in input variable x_i means that for all such $\mathbf{x}, \mathbf{x}'$ defined as above

$$f(\mathbf{x}') \geq f(\mathbf{x}) \tag{3}$$

Decreasing monotonicity is defined similarly.

[1]This is true assuming the model can implement f exactly. If this is not the case, satisfying a hint may lead to some loss in approximation ability, but is usually still beneficial on the balance.

Suppose we believe the target function f to be monotonic in all input variables- monotonically increasing in some and monotonically decreasing in others. We wish to define a single scalar measure of the degree to which a particular candidate function y violates this set of monotonicities.

One such natural measure, the one used in the experiments in Section IV, is defined in the following way: Let $\mathbf{x}$ be an input vector drawn from the input distribution. Let one input variable be chosen randomly from a uniform distribution over the set of input variables. Define a perturbation distribution, e.g., U[0,1], and draw δx_i from this distribution, where the subscript i indicates the input variable chosen. Define $\mathbf{x}'$ 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. We will call E_h the *monotonicity error* of y on the input pair $(\mathbf{x}, \mathbf{x}')$.

$$E_h = \begin{cases} 0 & y(\mathbf{x}') \geq y(\mathbf{x}) \\ (y(\mathbf{x}) - y(\mathbf{x}'))^2 & y(\mathbf{x}') < y(\mathbf{x}) \end{cases} \tag{6}$$

Our measure of y's violation of f's set of monotonicities is $E[E_h]$, where the expectation is taken with respect to random variables $\mathbf{x}$, i, and δx_i.

We believe that the best possible approximation to f given the architecture used is probably approximately monotonic. This belief may be quantified in a prior distribution over the candidate functions implementable by the architecture:

$$Pr(y) \propto e^{-\lambda E[E_h]} \tag{7}$$

This distribution represents the *a priori* probability density, or likelihood, assigned to a candidate function with a given level of monotonicity error. The normalization constant is unimportant and is neglected here. The probability that a function is the best possible approximation to f decreases exponentially with the increase in monotonicity error. λ is a positive constant which indicates how strong our bias is towards monotonic functions.

In addition to obeying prior information, the model should fit the data well. The two possible classes are denoted by 0 and 1, respectively. We take the network output y to represent the probability of class 1 conditioned on the observation of the input vector. Under this interpretation, the log likelihood of observing the data (normalized by the number of examples M) assuming that the network is the correct model is given by

$$\frac{1}{M} \sum_{m=1}^{M} c_m log(y_m) + (1 - c_m) log(1 - y_m) \tag{8}$$

where m is an index over the examples, c_m represents the class of example m, and y_m represents the network output on example m. We wish to pick the most probable model given the data. Equivalently, we may choose to maximize $log(P(model|data))$. Using Bayes' Theorem,

$$log(P(model|data)) \propto log(P(data|model) + log(P(model)) \tag{9}$$

$$= \frac{1}{M} \sum_{m=1}^{M} c_m log(y_m) + (1 - c_m) log(1 - y_m) - \lambda E[E_h] \tag{10}$$

The Bayesian prior leads to a familiar form of objective function, with the first term reflecting the desire to fit the data and a second term penalizing deviation from monotonicity.

3 Experimental Results

The credit card database was obtained via FTP from the machine learning database repository maintained by UC-Irvine [2]. The 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. The two class labels occur with nearly equal frequency; the split is 55%-45%.

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.

In all experiments, batch-mode backpropagation with a simple adaptive learning rate scheme was used [3]. 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. The most basic method tested was simply to train the network on all the training data for the full 5000 iterations. Another technique tried was to use a validation set to avoid overfitting. Training for all of these models was performed by maximizing (8). Finally, training the networks with the monotonicity constraints was performed, using an approximation to (10).

The linear perceptrons were trained for 300 batch mode iterations each, while the nonlinear networks were trained for 5000 batch mode iterations. 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 into 550 training examples and 116 test examples. 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 450 examples used for direct training and 100 validation examples. The classification error on the validation set was monitored over the course of the 5000 iterations, and the values of the network weights at the point of lowest validation error were chosen as the final values.

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.

Once the directions of monotonicity were determined, a two-layer network with 6 hidden units was trained by maximizing an approximation to the theoretical objective function (10) derived in the previous section:

$$\frac{1}{M}\sum_{m=1}^{M} c_m log(y_m) + (1-c_m)log(1-y_m) - \frac{\lambda}{N}\sum_{n=1}^{N} E_{h,n} \tag{11}$$

$E_{h,n}$ represents the network's monotonicity error on a particular pair of input vectors $\mathbf{x}, \mathbf{x}'$. Each pair was generated according to the method described in Section II. The input distribution was modelled as a joint gaussian with a covariance matrix estimated from the training data. The input variables were normalized to have zero mean and unit variance.

For each input variable, 500 pairs of vectors representing monotonicity in that variable were generated. This yielded a total of N=3000 hint example pairs. λ was chosen to be 5000. No optimization of λ was attempted; 5000 was chosen somewhat arbitrarily as simply a high value which would greatly penalize

[2] The database may be obtained as follows: ftp ics.uci.edu. cd pub/machine-learning-databases/credit-screening.

[3] If the previous iteration resulted in a increase in likelihood, the learning rate was increased by 3%. If the likelihood decreased, the learning rate was cut in half

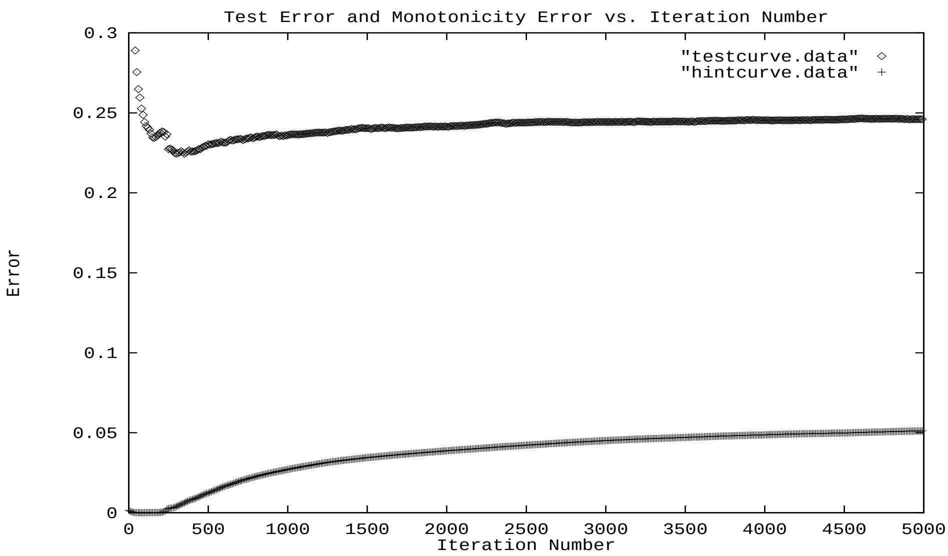

Figure 1: The violation of monotonicity tracks the overfitting occurring during training

non-monotonicity. 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 is a plot of out-of-sample classification rate (test error) and of monotonicity error vs. training time for the networks trained only on (8), 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: Performance of methods tested

The performance of each method is shown in table 1. 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 (nearly 2%) results from the introduction of monotonicity hints. Such an improvement could translate into a substantial increase in profit for a bank.

4 Conclusion

This paper has shown that monotonicity hints can significantly improve the performance of a neural network on a noisy binary classification task. It is worthwhile to note that the beneficial effect of imposing monotonicity does not necessarily imply that the target function is entirely monotonic. If there exist some non-monotonicities in the target function, then monotonicity hints may result in some decrease in the model's ability to implement this function. It may be, though, that this penalty is outweighed by the improved estimation of model parameters due to the decrease in model complexity. Therefore, the use of monotonicity hints probably should be considered in cases where the target function is thought to be at least roughly monotonic and the training examples are limited in number and noisy.

The reverse of the above point is also true: the target function may be strictly monotonic, and yet it may be best in some cases to enforce monotonicity in a model only approximately. Normally, it is not known whether the model being used has sufficient power to implement the target function exactly. Suppose the model does not have enough resources (e.g. not enough hidden units). Then there will be some discrepancy between the model's best possible approximation y^* and the target function f. From the monotonicity of f, it does not necessarily follow that y^* is *strictly* monotonic. The model may need to violate monotonicity slightly in order to do the best it can in approximating f. Enforcing monotonicity in the way presented here (i.e., "softly", with a penalty term) allows the model this freedom.

Future work may include the application of monotonicity hints to other real world problems and further investigations into the experimental details of enforcing the hints. An interesting theoretical question also arises: what is the capacity of the class of monotonic functions? The answer would provide a bound on the capacity of any particular model where monotonicity hints are enforced.

References

[1] S. Geman, E. Bienenstock & R. Doursat. "Neural Networks and the bias/variance dilemma" *Neural Computation*, vol. 4, pp. 1-58, 1992.

[2] Y. Abu-Mostafa. "Hints" *Neural Computation*, vol. 7, pp. 639-671, 1995.

[3] Y. Abu-Mostafa. "Hints and the VC Dimension" *Neural Computation*, vol. 4, pp. 278-288, 1993.

[4] P. Simard, Y. LeCun & J Denker. "Efficient Pattern Recognition Using a New Transformation Distance" *NIPS5*, pp. 50-58.

[5] Y. Abu-Mostafa. "Financial Market Applications of Learning from Hints" *Neural Networks in the Capital Markets*, A. Refenes, ed., pp. 221-232. Wiley, London, UK.