appears in Neural Computation, vol. 7, pp.639-671, MIT Press, July 1995

HINTS

Yaser S. Abu-Mostafa
California Institute of Technology
Pasadena, CA 91125, USA
e-mail: yaser@caltech.edu

The systematic use of hints in the learning-from-examples paradigm is the subject of this review. Hints are the properties of the target function that are known to us independently of the training examples. The use of hints is tantamount to combining rules and data in learning, and is compatible with different learning models, descent techniques, and regularization techniques. All the hints are represented to the learning process by virtual examples, and the training examples of the target function are treated on equal footing with the rest of the hints. A balance is achieved between the information provided by the different hints through the choice of objective functions and learning schedules. The Adaptive Minimization algorithm achieves this balance by relating the performance of the hints to the overall test error. On the theoretical side, the information value of hints is contrasted to the complexity value and related to the VC dimension. The application of hints in the forecasting of the very noisy foreign-exchange markets is illustrated.

1. INTRODUCTION

The context of this paper is learning from examples, where the learning process tries to recreate a target function using a set of input-output examples. *Hints* are the auxiliary information about the target function that can be used to guide the learning process [2]. The operative word here is *auxiliary.* There is quite a bit of information already contained in the input-output examples. There is also information reflected in the selection of the learning model (e.g., a neural network of a particular structure). If, in addition, we know some properties that further delimit the target function, we have hints. This paper reviews the theory, algorithms, and applications of how hints can be systematically incorporated in the learning process.

Hints can make a real difference in some applications. A case in point is financial forecasting [5]. Financial data is both nonstationary and extremely noisy. This limits the amount of relevant data that can be used for training, and limits the information content of such data. However, there are many hints about the behavior of financial markets that can be used to help the learning process. A hint as simple as symmetry of the foreign-exchange (FX) markets results in a statistically significant differential in performance as shown in figure 1. The plots show the averaged cumulative returns for the four major FX currencies over a sliding one-year test window, with and without the symmetry hint.

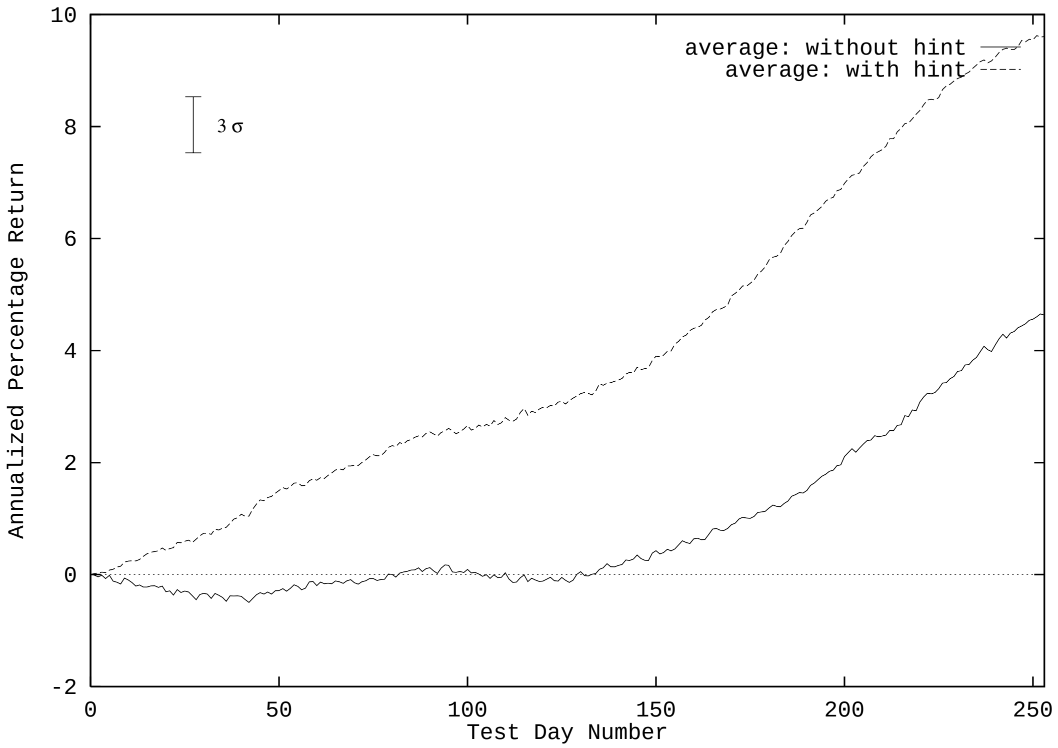

Figure 1: The impact of the symmetry hint on Foreign-Exchange rate forecasting

Just by analyzing the FX training data, one cannot deduce that the symmetry hint is valid. The hint is thus an auxiliary piece of information, telling the learning process something new. This is a double-edged sword because, by the same token, one cannot *verify* that the symmetry hint is valid just by analyzing the training data. A false hint, such as antisymmetry, can be asserted and used in the learning process equally easily. It is the performance after the hint is used that ultimately validates the hint. In the case of antisymmetry of the FX markets, figure 2 establishes that it is indeed a false hint. It may be possible, however, to *partially* detect or validate a hint using the training data. In those cases, the 'auxiliary' information of the hint is only incremental.

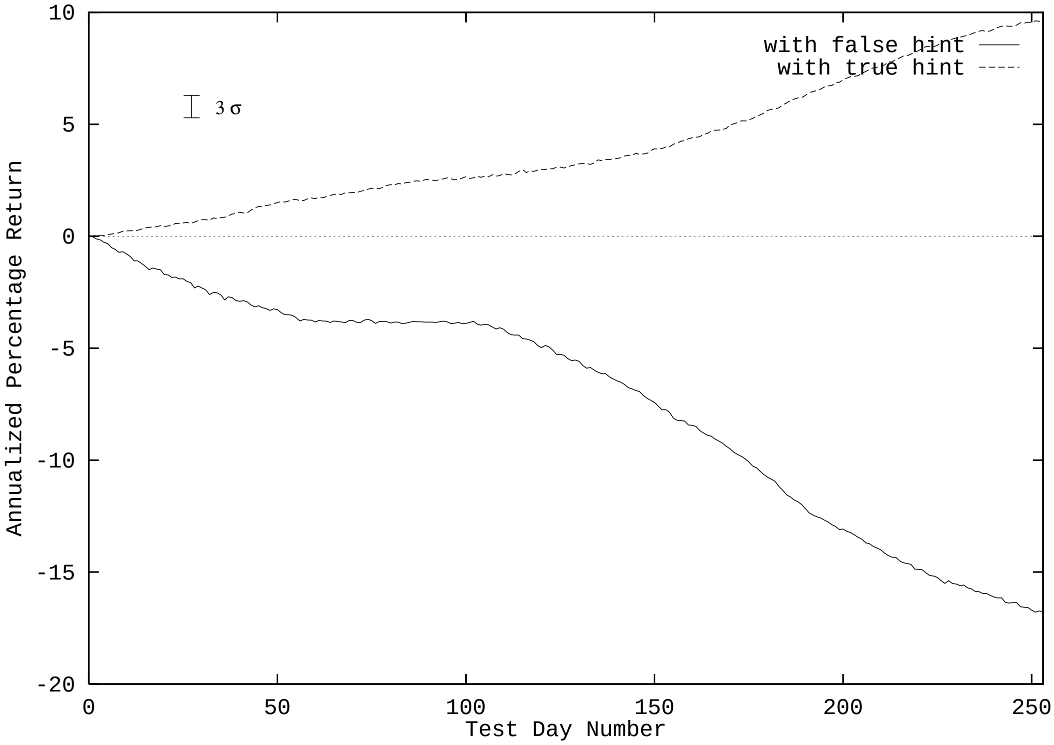

Figure 2: Comparison of the performance of the true hint versus a false hint

This brings us to a key point. The performance of learning from hints will only be as good as the hints we use. Valid hints that provide significant new information usually come from special expertise in the application domain, and from common-sense rules. The techniques we are reviewing here are not meant to *generate* hints in a given application. They only prescribe how to integrate the hints in the learning-from-examples paradigm once they are identified. There are 'information recycling' methods that are tantamount to the automated generation of hints from the training data, and these are not reviewed here.

The main purpose of using hints is to improve the generalization (out-of-sample) performance. As a constraint on the set of allowable solutions the learning process may settle in, the hint tends to worsen the training (in-sample) performance by excluding some solutions that might fit the training data better. This is obviously not a problem because, as the hint is a valid property of the target function, the excluded solutions disagree with the target function and correspond to 'fitting the noise' (overfitting the training data). In contrast with regularization techniques [6,24,33], which also constrain the allowable solutions to prevent overfitting, it is the *information* content of the hint that improves the out-of-sample performance. Figure 3 illustrates the difference. When the symmetry hint in FX is replaced by a hint which is uninformative but equally constraining ('noise' hint), the benefit of the hint is diminished.

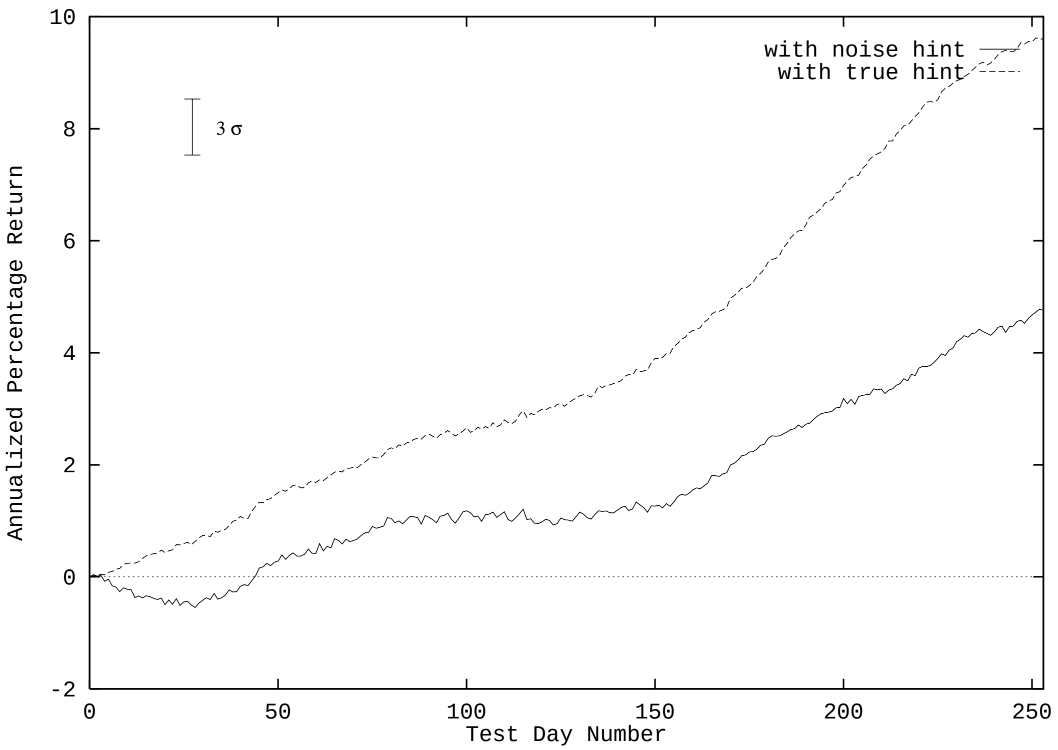

Figure 3: Comparison of the performance of the true hint versus a noise hint

Hints have other incidental effects on learning. Regularization is one of them. Even if the hint is not valid, its constraining role may improve generalization. Comparing the performance of the noise hint in figure 3 to that of no hint in figure 1, the regularization effect in this particular application is negligible. Another side effect of hints is a computational one. We observed in our experiments that the descent algorithm often had easier time finding a good minimum of the training error when we used hints. A more deliberate use of this effect is reported in [30], where a catalyst hint was used for the express purpose of avoiding local minima. Thus the hint was needed for its complexity value rather than its information value (see section 2). Out-of-sample performance was not at issue in this application since an unlimited supply of training examples from target function was readily available.

The applications that benefit the most from hints are those in which the training examples are limited (costly, or outright limited in number), and those in which the information in the training examples is limited (noisy examples, or examples that do not depict some aspects of the target function). Like regularization, hints would not be needed if an unlimited supply of proper examples (and unlimited computation) could be secured.

There are different types of hints that are common to many applications. Invariance hints [12,16,19,23] are the most common type in pattern recognition applications. Such a hint asserts that the target function is invariant under certain transformations of the input. Monotonicity hints [4] are common in 'expert system' applications such as credit rating and medical diagnosis.

The two major steps for using hints in learning from examples are the representation of hints by virtual examples and the incorporation of hints in the objective function. A *virtual example* is for the hint what a training example is for the target function. It is a sample of the information provided by the hint. Figure 4 shows a virtual example of the invariance hint for handwritten characters under a composite transformation of elastic deformation [17], scaling, shift, and rotation. The example takes the form of a pair of inputs that are transformed versions of each other. A virtual example does not provide the value of the target function (the identity of the character ξ in this case). It only asserts that the identity is the same for both versions of the character.

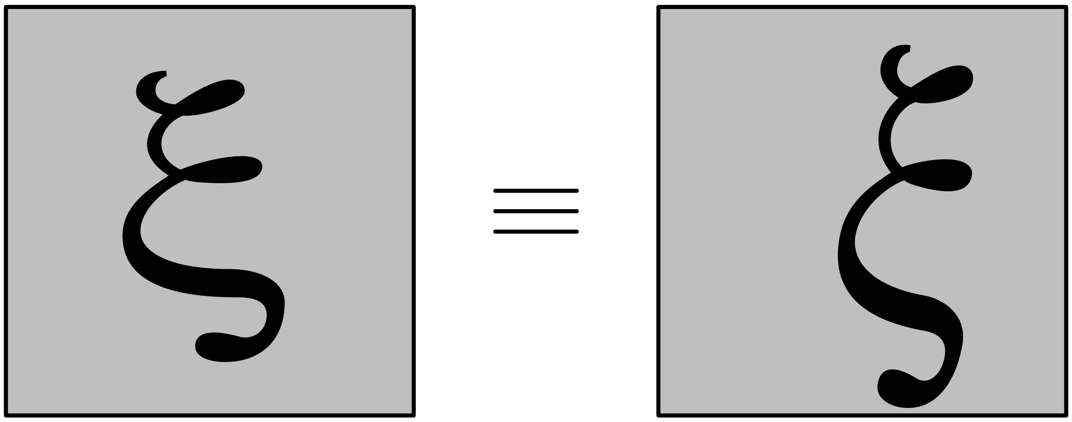

Figure 4: Virtual Example of an Invariance Hint

After the hint is represented by virtual examples, we can measure how well it has been learned by gauging how well the system is performing on a batch of these examples. This error measure is the way the performance of the hint is expressed to the objective function. The choice of an objective function is the second step in our method. Without hints, the objective function is usually just the error on the training examples. With the hints, we not only want to minimize the error on the training examples, but also the error on the different hints. The simultaneous minimization of errors gives rise to the issue of balance. What is the scalar objective function that gives each of these error measures its due weight? This weight determines how often each hint is scheduled in the learning process. In section 4, we will discuss Adaptive Minimization that decides these weights by relating the different error measures to the overall test error.

Not all applications have obvious hints to spare. One practical way of extracting subtle hints from the experts in a given application (e.g., traders in a financial market) is to create a system without using hints, and, when the experts disagree with its output, ask them to articulate why they disagree. The hints created this way are inherently auxiliary since they were not exhibited in the system output. Another practical issue is that it is often tricky to ascertain whether a hint is strictly valid, or just 'approximately' valid. Some of the more useful hints are *soft hints* which hold most of the time, but not all the time. The use of error measures to represent different hints allows us to incorporate soft hints in the same paradigm by not requiring their error to go all the way to zero.

The idea of using auxiliary information about the target function to help the learning process is clearly a basic one, and has been used in the literature under different names (hints, side information, heuristics, prior knowledge, explicit rules, to name a few). In many instances, the information is used on a case-by-case basis to guide the selection of a suitable learning model. In this paper, we are only reviewing the *systematic* methods for using hints as part of the regular learning paradigms. Such methods are particularly important because hints are heterogeneous in nature, and do not lend themselves to a standard implementation in most cases.

The outline of the paper is as follows. We start by discussing the theoretical background in section 2. The information value and the complexity value of hints are defined. The VC dimension is used to quantify the information value, and a numerical example is given. Section 3 discusses the representation of hints using virtual examples, and the resulting error measures. The representation is carried out for common types of hints. Section 4 addresses the objective functions and the learning schedules in terms of the error measures on different hints. Adaptive Minimization is discussed and simple estimates of its objective function are included. Finally, the application of hints to the FX markets is detailed in section 5.

2. THEORETICAL ISSUES

In this section, we discuss the theoretical aspects of learning from hints. We contrast the information value with the complexity value of hints, and quantify the information value in terms of the VC dimension. We first introduce the definitions and notation.

2.1) Basic Setup:

We use the usual setup for learning from examples. The *environment* X is the set on which the target 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 input-output examples of (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 common forms of the error measure are

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

and

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

where $\Pr[.]$ denotes the probability of an event, and $\mathcal{E}[.]$ denotes the expected value of a random variable. 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.

If the learning model is a parameterized set of hypotheses with real-valued parameters (e.g., an analog neural network), we will assume that E is well-behaved as a function of the parameters when we use derivative-based descent techniques. We make the same assumption about the error measures that will be introduced for the hints.

The training examples are generated from the target function f by picking a number of points $x_1, \cdots, x_N$ from X (usually independently according to the probability distribution P). The values of f on these points are given (noiseless case). Thus, the input to the learning process is the set of examples

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

and these examples are used to guide the search for a good hypothesis. 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 denote additional properties of f that are known to us. The training error on the examples of f will be denoted by E_0, while the error measures on the different hints will be denoted by $E_1, \cdots, E_M$.

2.2) Information versus Complexity:

Since the goal of hints is to help learning from examples, they address the problems that the learning process may have. There are two such problems in learning from examples:

1. Do the examples convey enough information to replicate the target function?
2. Is there a speedy way of replicating the function using the examples?

These questions contrast the roles of information and complexity in learning [1]. The information question is manifested in the generalization error while the complexity question is manifested in the computation time. While the two questions share some ground, they are conceptually and technically different. Without sufficient information, no algorithm slow or fast can produce a good hypothesis. However, sufficient information is of little use if the computational task of producing a good hypothesis is intractable [20].

A hint may be valuable to the learning process in two ways [2]. It may reduce the number of hypotheses that are candidates to replicate f (information value), and it may reduce the amount of computation needed to find the right hypothesis (complexity value). The contrast between the information value and the complexity value of a hint is illustrated in the following example.

A target function f is being learned by a neural network with K weights, labeled $w_1, w_2, \cdots, w_K$ for simplicity. Which of the following two hints is more valuable to the learning process? (Both hints are artificial, and are meant only for illustration)

1. f can be implemented using the network with w_1 set to zero.

2. f can be implemented using the network with $w_1, \cdots, w_K$ constrained by the two conditions $\sum_{k=1}^{K} w_k = 0$ and $\sum_{k=1}^{K} w_k^3 = 0$.

If we look at the information value, the second hint is more valuable because it reduces the K 'degrees of freedom' of the network by 2, while the first hint reduces them only by 1. The situation is reversed when it comes to complexity value. The second hint is worse than no hint at all, since it adds two constraints to the otherwise unconstrained optimization problem. In contrast, the first hint has a positive effect, since the algorithm can fix $w_1 = 0$, hence deal with a smaller computational problem ($K - 1$ parameters instead of K parameters).

Most hints are used for their information value. However, the catalyst hint was used in [30] for its complexity value to help a network learn the concept of medium height. The hint itself was the concept of tallness, a monotonic version of the other concept. As a result of using the hint, the network had easier time converging to the solution without getting stuck in local minima. There was an unlimited supply of training examples in this case, so information was not an issue.

2.3) New VC Dimensions:

The VC dimension [9,31] is an established tool for analyzing the question of information in learning from examples. Simply stated, the VC dimension $VC(G)$ furnishes an upper bound on the number of examples needed by the learning process that starts with a learning model G, where G is formally a set of hypotheses about what f may be. The examples guide the search for a hypothesis $g \in G$ that is a good replica of f. Since f is unknown to begin with, we start with a relatively big set of hypotheses G to maximize our chances of finding a good approximation of f among them. However, the bigger G is, the more examples of f we need to pinpoint a good hypothesis. This is reflected in a bigger value of $VC(G)$.

When a hint is introduced, the VC dimension is affected. Since the hint is a valid property of f, we can use it as a litmus test to weed out bad g's thus shrinking G

without losing good hypotheses. This leads to two new VC dimensions [3]:

1. The VC dimension provides an estimate for the number of examples needed to learn f, and since a hint H reduces the number of examples needed, a smaller 'VC dimension given the hint', $VC(G|H)$, emerges.

2. If H itself is represented to the learning process by virtual examples, we can ask how many examples are needed to learn the hint. This leads to a generalization of the VC dimension to cover examples of the hint as well as examples of the function. A new VC dimension for the hint, $VC(G;H)$, emerges.

We start with a brief explanation of how the original VC dimension is defined. We have the same setup for learning from examples: The environment X and the target function $f : X \to \{0,1\}$ (restricted to binary values here). The goal is to produce a hypothesis $g : X \to \{0,1\}$ (also restricted to binary values) that approximates f. To do this, the learning process uses a set of training examples $(x_1, f(x_1))$; ... ; $(x_N, f(x_N))$ of f. We use the probability distribution P on the environment X to generate the examples. Each example $(x, f(x))$ is picked independently according to $P(x)$. The hypothesis g that results from the learning process is considered a good approximation of f if the probability (w.r.t. $P(x)$) that $g(x) \neq f(x)$ is small. The learning process should have a high probability of producing a good approximation of f when a sufficient number of examples is provided. The VC dimension helps determine what is 'sufficient'.

Here is how it works. Let $\pi_g = \Pr[g(x) = f(x)]$ be the probability of agreement between g and f $(= 1 - E(g,f))$. We wish to pick a hypothesis g that has $\pi_g \approx 1$. However, f is unknown and thus we don't know the values of these probabilities. Since f is represented by examples, we can compute the frequency of agreement between each g and f *on the examples* and base our choice of g on the frequencies instead of the actual probabilities. Let hypothesis g agree with f on a fraction ν_g of the examples. We pick a hypothesis that has $\nu_g \approx 1$. The VC inequality asserts that the values of ν_g's will be close to π_g's. Specifically,

$$\Pr\left[\sup_{g \in G} |\nu_g - \pi_g| > \epsilon\right] \;\le\; 4m(2N)e^{-\epsilon^2 N/8},$$

where 'sup' denotes the supremum, and m is the *growth function* of G. $m(N)$ is the maximum number of different binary vectors $g(x_1)\cdots g(x_N)$ that can be generated by varying g over G while keeping $x_1, \cdots, x_N \in X$ fixed. Clearly, $m(N) \le 2^N$ for all N. The VC dimension $VC(G)$ is defined as the smallest N for which $m(N) < 2^N$. When

G has a finite VC dimension, $VC(G) = d$, the growth function $m(N)$ can be bounded by

$$m(N) \ < \ \sum_{i=0}^{d} \binom{N}{i} \ \leq \ N^d + 1$$

This estimate can be substituted in the VC inequality, and the RHS of the inequality becomes arbitrarily small for sufficiently large N. This means that it is almost certain that each ν_g will be approximately the same as the corresponding π_g. This is the rationale for considering N examples sufficient to learn f. We can afford to base our choice of hypothesis on ν_g as calculated from the examples, because it is approximately the same as π_g. How large N needs to be to achieve a certain degree of approximation is affected by the value of the VC dimension.

The same ideas can be used in deriving the new VC dimensions $VC(G|H)$ and $VC(G;H)$ when the hint H is introduced. For instance, let H be an invariance hint formalized by the partition

$$X = \bigcup_{\lambda} X_\lambda$$

of the environment X into the invariance classes X_λ, where λ is an index. Within each class X_λ, the value of f is constant. In other words, $x, x' \in X_\lambda$ implies that $f(x) = f(x')$.

Some invariance hints are 'strong' and others are 'weak', and this is reflected in the partition $X = \bigcup_\lambda X_\lambda$. The finer the partition, the weaker the hint. For instance, if each X_λ contains a single point, the hint is extremely weak (actually useless) since the information that $x, x' \in X_\lambda$ implies that $f(x) = f(x')$ tells us nothing new as x and x' are the same point in this case. On the other extreme, if there is a single X_λ that contains all the points ($X_\lambda = X$), the hint is extremely strong as it forces f to be constant over X (either $f = 1$ or $f = 0$). Practical hints, such as scale invariance and shift invariance, lie between these two extremes. The strength or weakness of the hint is reflected in the quantities $VC(G|H)$ and $VC(G;H)$.

$VC(G|H)$ is defined as follows. If H is given by the partition $X = \bigcup_\lambda X_\lambda$, each hypothesis $g \in G$ either satisfies H or else does not satisfy it. Satisfying H means that whenever $x, x' \in X_\lambda$, then $g(x) = g(x')$. The set of hypotheses that satisfy H is $\hat{G}$

$$\hat{G} = \{g \in G \mid x, x' \in X_\lambda \Rightarrow g(x) = g(x')\}$$

$\hat{G}$ is a set of hypotheses and, as such, has a VC dimension of its own. This is the basis for defining the VC dimension of G given H

$$VC(G|H) = VC(\hat{G})$$

Since $\hat{G} \subseteq G$, it follows that $VC(G|H) \leq VC(G)$. Non-trivial hints lead to a significant reduction from G to $\hat{G}$, resulting in $VC(G|H) < VC(G)$.

$VC(G|H)$ replaces $VC(G)$ after the hint is learned. Without the hint, $VC(G)$ provides an estimate for the number of examples needed to learn f. With the hint, $VC(G|H)$ provides a new estimate for the number of examples. This estimate is valid regardless of the mechanism for learning the hint, as long as it is completely learned. If, however, the hint is only *partially* learned (which means that some g's that do not strictly satisfy the invariance are still allowed), the effective VC dimension lies between $VC(G)$ and $VC(G|H)$.

The other VC dimension $VC(G;H)$ arises when we represent the hint by virtual examples. If we take the invariance hint specified by $X = \bigcup_\lambda X_\lambda$, a virtual example would be '$f(x) = f(x')$', where x and x' belong to the same invariance class. In other words, the example is the pair (x, x') that belong to the same X_λ.

Examples of the hint, like examples of the function, are generated according to a probability distribution. One way to generate (x, x') is to pick x from X according to the probability distribution $P(x)$, then pick x' from X_λ (the invariance class that contains x) according to the conditional probability distribution $P(x'|X_\lambda)$. A sequence of N examples $(x_1, x_1'); (x_2, x_2'); \cdots; (x_N, x_N')$ would be generated in the same way, independently from pair to pair.

This leads to the definition of $VC(G;H)$. The VC inequality is used to estimate how well f is learned. We wish to use the same inequality to estimate how well H is learned. To do this, we transform the situation from hints to functions. This calls for definitions of new $\mathbf{X}$, $\mathbf{P}$, $\mathbf{G}$, and $\mathbf{f}$.

Let H be the invariance hint $X = \bigcup_\lambda X_\lambda$. The new environment is defined by

$$\mathbf{X} = \bigcup_\lambda X_\lambda^2$$

(pairs of points coming from the same invariance class) with the probability distribution described above

$$\mathbf{P}(x, x') = P(x)P(x'|X_\lambda)$$

where X_λ is the class that contains x (hence contains x'). The new set of hypotheses $\mathbf{G}$, defined on the environment $\mathbf{X}$, contains a hypothesis $\mathbf{g}$ for every hypothesis $g \in G$ such that

$$\mathbf{g}(x, x') = \begin{cases} 1 & g(x) = g(x') \\ 0 & g(x) \neq g(x') \end{cases}$$

and the function to be 'learned' is

$$\mathbf{f}(x, x') = 1$$

The VC dimension of the set of hypotheses $\mathbf{G}$ is the basis for defining a VC dimension for the hint.

$$VC(G; H) = VC(\mathbf{G})$$

$VC(G; H)$ depends on both G and H since $\mathbf{G}$ is based on G and the new environment $\mathbf{X}$ (which in turn depends on H).

As in the case of the set G and its growth function $m(N)$, the VC dimension $VC(G; H) = VC(\mathbf{G})$ is defined based on the growth function $\mathbf{m}(N)$ of the set $\mathbf{G}$. $\mathbf{m}(N)$ is the maximum number of different binary vectors that can be obtained by applying the $\mathbf{g}$'s to (fixed but arbitrary) N examples $(x_1, x'_1); (x_2, x'_2); \cdots; (x_N, x'_N)$. $VC(G; H)$ is the smallest N for which $\mathbf{m}(N) < 2^N$.

The value of $VC(G; H)$ will differ from hint to hint. Consider our two extreme examples of weak and strong hints. The weak hint has $VC(G; H)$ as small as 1 since each g always agrees with each example of the hint (hence every $\mathbf{g}$ is the constant 1, and $\mathbf{m}(N) = 1$ for all N). The strong hint has $VC(G; H)$ as large as it can be. How large is that? In [14], it is shown that for any invariance hint H,

$$VC(G; H) \quad < \quad 5 \ VC(G)$$

The argument goes as follows. For each pattern generated by the g's on

$$x_1 \ , \ x'_1 \ , \ x_2 \ , \ x'_2 \ , \ \ \cdots \ \ , \ x_N \ , \ x'_N$$

there is at most one distinct pattern generated by the $\mathbf{g}$'s on

$$(x_1, x'_1) \ ; \ (x_2, x'_2) \ ; \ \ \cdots \ \ ; \ (x_N, x'_N)$$

because $\mathbf{g}(x_n, x'_n)$ is uniquely determined by $g(x_n)$ and $g(x'_n)$. Therefore,

$$\mathbf{m}(N) \leq m(2N)$$

If $VC(G) = d$, we can use Chernoff bounds [13] to estimate $m(2N)$ for $N \geq d$ as follows

$$\begin{aligned} m(2N) &< \sum_{i=0}^{d} \binom{2N}{i} \\ &\leq 2^{\mathcal{H}(\frac{d}{2N}) \times 2N} \end{aligned}$$

where $\mathcal{H}(\theta) = -\theta \log_2 \theta - (1-\theta)\log_2(1-\theta)$ is the binary entropy function. Therefore, once $\mathcal{H}(\frac{d}{2N}) \leq \frac{1}{2}$, $\mathbf{m}(N)$ will be less than 2^N and N must have reached, or exceeded, the VC dimension of $\mathbf{G}$. This happens at $N/d < 5$.

In many cases, the smaller $VC(G|H)$ is, the larger $VC(G;H)$ will be, and vice versa. Strong hints generally result in a small value of $VC(G|H)$ and a large value of $VC(G;H)$, while weak hints result in the opposite situation. The similarity with the average mutual information $I(X;Y)$ and the conditional entropy $H(X|Y)$ in information theory [11] is the reason for choosing this notation for the various VC dimensions.

2.4) Numerical Case:

To illustrate the numerical values of the new VC dimensions, we consider a simple case in which a small neural network (figure 5) learns a binary target function that is both even-valued (H_1) and invariant under cyclic shift (H_2). The network has 8 inputs, one hidden layer with 2 neurons, and one output neuron. The rule of thumb for the VC dimension of neural networks is that it is approximately the same as the number of real-valued parameters in the network (independent weights and thresholds). We will therefore calculate this number for the network before and after it is constrained by the different hints to get an estimate for $VC(G)$, $VC(G|H_1)$, $VC(G|H_2)$, $VC(G|H_1H_2)$. In each case, we consider the combination of weights and thresholds that maximizes the number of free parameters.

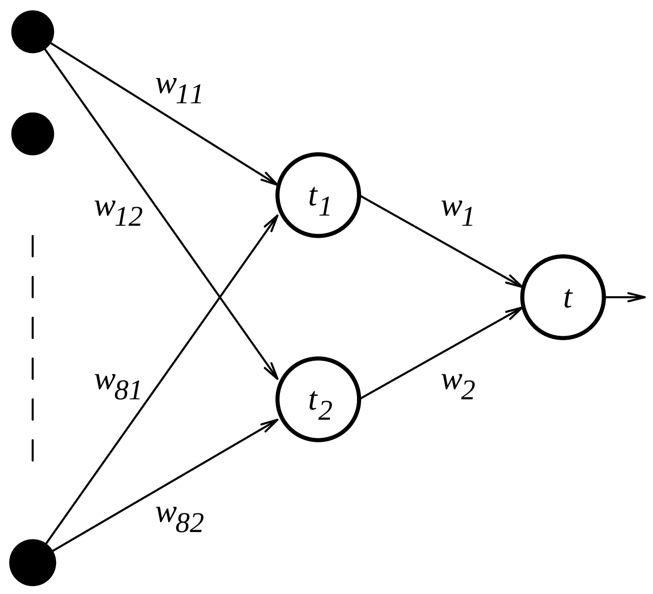

Figure 5: An 8-2-1 Neural Network

No Hints: The number of weights is $8 \times 2 + 2 = 18$ plus 3 thresholds. There are no constraints, therefore

$$VC(G) \approx 21$$

Evenness: To implement a general even function, the two hidden units need to be dual; $w_{11} = -w_{12}$, $w_{21} = -w_{22}$, $\cdots$, $w_{81} = -w_{82}$, $t_1 = t_2$, $w_1 = w_2$. Therefore, the number of free parameters is $8 + 1 + 1 + 1$, hence

$$VC(G|H_1) \approx 11$$

Cyclic Shift: To implement a general function which is invariant under cyclic shift using the maximum number of free parameters, each hidden neuron must have constant weights; $w_{11} = w_{21} = \cdots = w_{81}$ and $w_{12} = w_{22} = \cdots = w_{82}$. Therefore, the number of free parameters is $1 + 1 + 2 + 2 + 1$, hence

$$VC(G|H_2) \approx 7$$

Notice that $VC(G|H_2) < VC(G|H_1)$ which is consistent with the intuition that cyclic shift is a stronger hint than evenness. Notice also that, for the 8-2-1 network, the constraint on the network would be the same if H_2 was invariance under permutation of inputs or invariance under constant component sum of inputs. With a different network, these hints can result in different values for $VC(G|H)$.

Both Hints: To implement a general even function which is also invariant under cyclic shift, we have the conjunction of the constraints on the two hidden neurons. Thus, $w_{11} = w_{21} = \cdots = w_{81} = -w_{12} = -w_{22} = \cdots = -w_{82}$. Also, $t_1 = t_2$ and $w_1 = w_2$. Therefore, the number of free parameters is $1 + 1 + 1 + 1$, hence

$$VC(G|H_1 H_2) \approx 4$$

which is significantly less than $VC(G)$, the VC dimension without hints.

Figure 6 shows the experimental values of the corresponding generalization error in all 4 cases. The training error, which is not shown, always goes to zero. The 4 curves show the test error without hints, with H_1, with H_2, and with both hints, versus the number of learning iterations. Since the number of training examples of f was the same in all 4 cases, the final test error more or less agrees with the above numerical

. Figure 6: The test error with combinations of hints

estimates and the rule of thumb that the generalization error is proportional to the VC dimension.

3. REPRESENTATION OF HINTS

In order to utilize hints in learning, we need to express them in a way that the learning algorithm would understand. The main step is to represent each hint by *virtual examples.* This enables the learning algorithm to process the hints in the same way it processes the training examples of f. The virtual examples give rise to error measures $E_1, E_2, \cdots, E_M$ that gauge the performance on the different hints, the same way E_0 gauges the performance on the training examples.

3.1) Virtual Examples:

Virtual examples were introduced in [2] as a means of representing a given hint, independently of the target function and the other hints. Duplicate examples, on the other hand, provide another way of representing certain types of hints by expanding the existing set of training examples, and will be discussed in **3.3)**.

To generate virtual examples, we need to break the information of the hint into small pieces. For illustration, suppose that H_m asserts that

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

is an *odd* function. A virtual 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 quantify how much g disagrees with this example. This is done through an 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., $g(-x) = -g(x)$). e_m can be handled by descent techniques the same way the error on an example of f is handled. For instance, the gradient of e_m is given by

$$\frac{\partial e_m}{\partial w} \;=\; \frac{\partial}{\partial w}(g(x) + g(-x))^2 \;=\; 2(g(x) + g(-x))\left(\frac{\partial g(x)}{\partial w} + \frac{\partial g(-x)}{\partial w}\right)$$

which can be implemented using two iterations of backpropagation [29].

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 the error measure 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 not unique. Neither is the form of the virtual examples nor the choice of the error measure. Therefore, E_m will depend on how we choose these components of the representation. Our choice is 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 valid for the target function f (recall that E is the error between g and f w.r.t. the original probability distribution P on the environment X). The condition is not necessary for *soft* hints, i.e., hints that are only 'approximately' valid.

To see how this condition can make a difference, 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, which means that 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 impose a stronger condition than the hint itself since we already have $E_m = 0$ when $g = f$.

On the practical side, there are other properties of virtual examples 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.

3.2) Types of Hints:

In what follows, we illustrate the representation of hints by virtual examples for some common 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 a virtual 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.

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, a virtual 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 $(x_1, f(x_1)), \cdots, (x_N, f(x_N))$, *the examples hint* asserts that these are the correct values of f at the particular points $x_1, \cdots, x_N$. Now, to generate an 'example' of this hint, we independently pick a number n from 1 to N and use the corresponding $(x_n, f(x_n))$ (figure 7). The error associated with this example is e_0 (we use the convention that $m = 0$ for the examples hint)

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

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

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

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

Another type of hint related to the examples 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

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

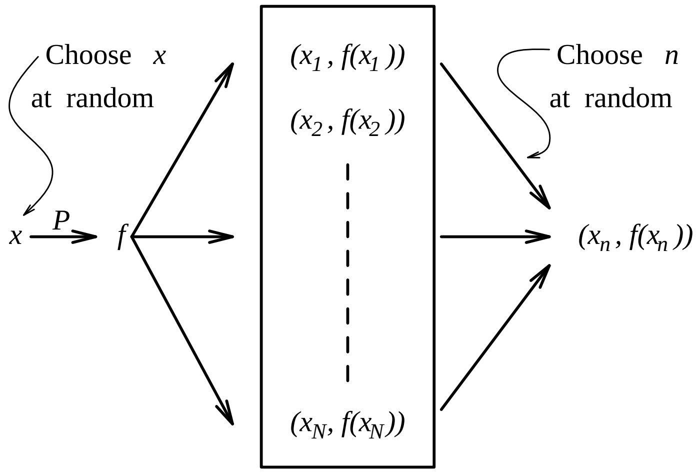

Figure 7: Examples of the function as a hint

When a new type of hint is identified in a given application, it should also be expressed in terms of virtual examples. The resulting error measure E_m will represent the hint in the learning process.

3.3) Duplicate Examples:

Duplicate examples are perhaps the easiest way to use certain types of hints, most notably invariance hints. If we start with a set of training examples from the target function f

$$(x_1, f(x_1)) \ , \ (x_2, f(x_2)) \ , \ \cdots, (x_N, f(x_N))$$

and then assert that f is invariant under some transformation of x into x', it follows that we also know the value of f on $x'_1, x'_2, \cdots, x'_N$. In effect we have a duplicate set of training examples

$$(x'_1, f(x'_1)) \ , \ (x'_2, f(x'_2)) \ , \ \cdots, (x'_N, f(x'_N))$$

where $f(x'_n) = f(x_n)$, that can be used along with the original set. For instance, duplicate examples in the form of new 2D views of a 3D object are generated in [28] based on existing prototypes.

When duplicate examples are used to represent a hint, the rest of the learning machinery is already in place. The training error E_0 can still be used as the objective function, with the training set now consisting of the original examples and the duplicate examples. The duplication process 'inherits' the probability distribution that was used to generate the original examples, which is usually the target distribution P. A balance, of sorts, is automatically maintained between the hint and training examples since both are learned through the same set of examples. The same software for learning from examples can be used unaltered.

On the other hand, there are two main disadvantages of duplicate examples that give an edge to virtual examples. The first one is the generalization error. The duplicate examples are fixed in terms of the original training set. Even if the training error goes to zero, this may not generalize to new examples. In effect, duplicate examples represent a restricted version of the hint. Virtual examples, on the other hand, are unlimited in number and hence do not have a generalization problem. The other problem is the linkage between the training error and the hint error. For duplicate examples, the two errors are entwined together in the new E_0. For virtual examples, E_0 and E_m are separate entities and each can be emphasized or deemphasized at will in the learning process. This 'separation of concerns' enables us to design our objective functions and learning schedules freely.

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. This leads to error measures $E_0, E_1, \cdots, E_M$ for the hints that are ready for incorporation in the objective function of a learning-from-examples algorithm.

4. OBJECTIVE FUNCTIONS

When hints are available in a learning situation, the objective function to be optimized by the learning algorithm is no longer confined to E_0 (the error on the training examples of f). This section addresses how to combine E_0 with the different hints to create a new objective function.

4.1) Adaptive Minimization:

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 $H_0, H_1, \cdots, H_M$ is to reach a hypothesis g for which all the error measures $E_0, E_1, \cdots, E_M$ are zeros.

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 will 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. This is the idea behind *Adaptive Minimization*.

Given $E_0, E_1, \cdots, E_M$, we form an estimate $\hat{E}$ of the actual error E

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

and use it as the objective function to be minimized. This estimate of E becomes the common thread that balances between the error on the different hints. The formula for $\hat{E}$ expresses the impact of each E_m on the ultimate performance. Such a formula is of theoretical interest in its own right.

$\hat{E}$ is minimized by the learning algorithm. For instance, if backpropagation is used, the gradient will be

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

which means that regular backpropagation can be used on each of the hints, with $\frac{\partial \hat{E}}{\partial E_m}$ used as the 'weight' for hint H_m. Equivalently, a batch of examples from the different hints would be used with a number of examples from H_m in proportion to $\frac{\partial \hat{E}}{\partial E_m}$. This idea is discussed further when we talk about schedules in **4.3)**.

4.2) Simple Estimates:

In [10], a simple formula for $\hat{E}(E_0, \cdots, E_M)$ is derived and tested for the case of a binary target function $f : \mathcal{R}^n \rightarrow \{0, 1\}$ that has two invariance hints. The learning model is a sigmoidal neural network, $g : \mathcal{R}^n \rightarrow [0, 1]$. The difference between f and g is viewed as a 'noise' function n:

$$n(x) = |f(x) - g(x)| = \begin{cases} 1 - g(x), & \text{if } f(x) = 1; \\ g(x), & \text{if } f(x) = 0. \end{cases}$$

Let μ and σ^2 be the mean and variance of $n(x)$. In terms of μ and σ^2, the error measure $E(g, f)$ is given by

$$E = \mathcal{E}((f(x) - g(x))^2) = \mathcal{E}(n^2(x)) = \mu^2 + \sigma^2$$

Similarly, the error on each of the two invariance hints is given by:

$$E_m = \mathcal{E}((g(x) - g(x'))^2) = \mathcal{E}((n(x) - n(x'))^2) = 2\sigma^2$$

assuming that $n(x)$ and $n(x')$ are independent random variables. Given the training examples, one can obtain a direct estimate of μ

$$[\mu] = \frac{1}{N_0} \sum_{i=1}^{N_0} |f(x_i) - g(x_i)|$$

and, combining this estimate with E_0, E_1, and E_2, one can get an estimate of σ^2

$$[\sigma^2] = \frac{2(E_0 - [\mu]^2) + E_1 + E_2}{6}$$

Finally, we get an estimate of E, based solely on the training examples of f and the virtual examples of the hints, by combining $[\mu]$ and $[\sigma^2]$

$$\hat{E} = [\mu]^2 + [\sigma^2]$$

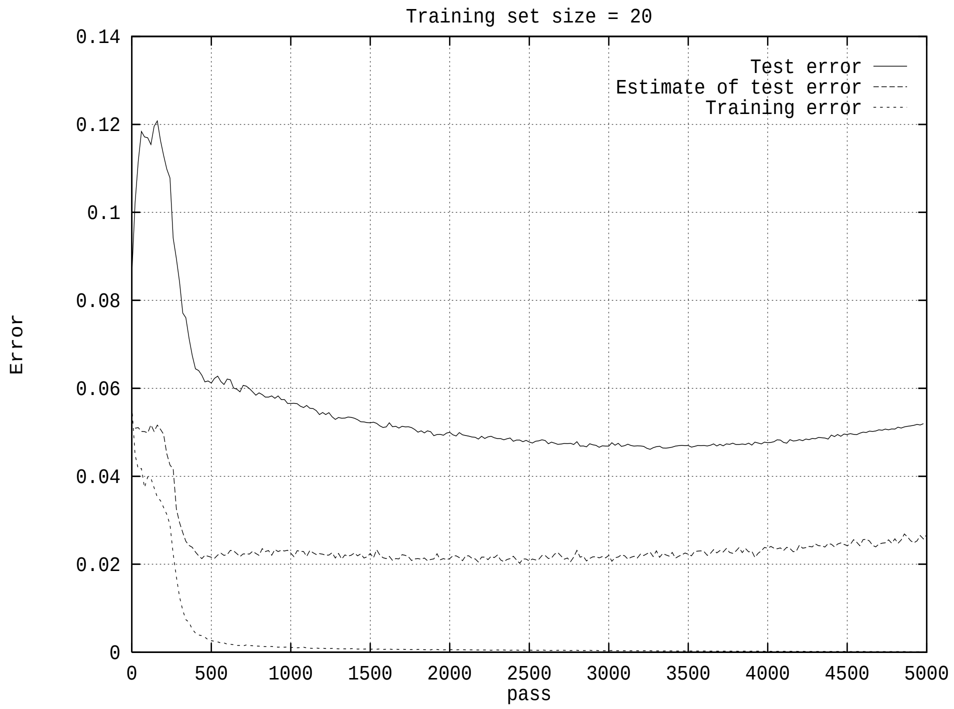

Figure 8: The error estimate in the case of overlearning

Figures 8,9 illustrate the performance of this estimate in two cases, one where overlearning occurs and the other where it doesn't. The figures show the pass number of regular backpropagation versus the training error (E_0), test error (E), and the estimate of the test error ($\hat{E}$). Notice that $\hat{E}$ is closer to the actual E than E_0 is (E_0 is the *de facto* estimate of E in the absence of hints). $\hat{E}$ is roughly monotonic in E and, as seen in figure 8, exhibits the same increase due to overlearning that E exhibits. The significant difference between E and $\hat{E}$ is in the form of (almost) a constant. However, constants do not affect descent operations. Thus, $\hat{E}$ provides a better objective function than E_0, even with the simplifying assumptions made.

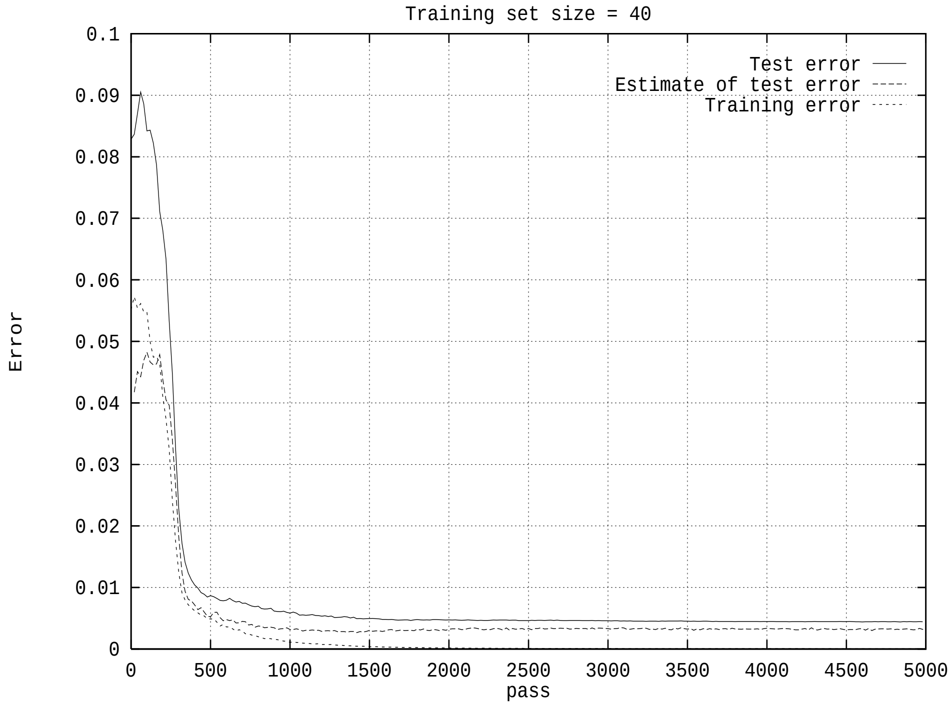

Figure 9: The error estimate without overlearning

4.3) Schedules:

The question of objective functions can be posed as a scheduling question: If we are simultaneously minimizing the interrelated quantities $E_0, \cdots, E_M$, how do we schedule which quantity to minimize at which step? To start with, let us explore how simultaneous minimization of a number of quantities is done. Perhaps the most common method is that of *penalty functions* [35]. In order to minimize $E_0, E_1, \cdots, E_M$, 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 reflects 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.

In the case of hints, 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 a virtual example of the hint). Instead, we will *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 n^{th} example. Consider a batch of examples consisting of N_0 examples of H_0, N_1 examples of H_1, $\cdots$, 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.

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

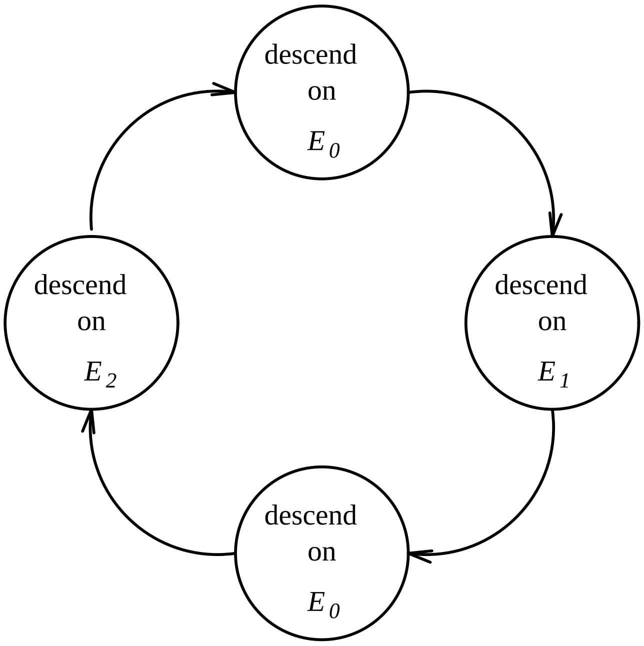

Figure 10: A fixed schedule for learning from hints

the same batch. For instance, as we discussed before, Adaptive Minimization can be implemented using backpropagation on a mixed batch where hint H_m is represented by a number of examples proportional to $\frac{\partial \hat{E}}{\partial E_m}$. 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 10 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 difficult to ascertain a direct translation from mixed batches to uniform batches.

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 $m = 0, 1, \cdots, M$) 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. Here are some simple schedules.

Simple Rotation: This is the simplest possible schedule that tries to balance between the hints. It is a fixed schedule that rotates between $H_0, H_1, \cdots, H_M$. 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 α_m. The choice of the α_m's can achieve balance by emphasizing the hints that are more important or harder to learn. The schedule of figure 10 is a weighted rotation with $\alpha_0 = 0.5$ and $\alpha_1 = \alpha_2 = 0.25$.

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 $\alpha_m E_m$. The choice of the α_m's can achieve balance by making up for disparities between the numerical ranges 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. Adaptive Minimization answers the question by relating the E_m's to the actual error E. Here is the uniform-batch version of Adaptive Minimization

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

$$\begin{gathered}\hat{E}(\bullet, E_1, E_2, \cdots, E_M)\\ \hat{E}(E_0, \bullet, E_2, \cdots, E_M)\\ \hat{E}(E_0, E_1, \bullet, \cdots, E_M)\\ \cdots\\ \hat{E}(E_0, E_1, E_2, \cdots, \bullet)\end{gathered}$$

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

The idea is that if the absence of E_m resulted in the most optimistic view of E, then E_m carries the worst news and, hence, requires immediate attention. One way of expressing what the AM schedule does is that it automatically pays more attention to hints that are more important (their impact on $\hat{E}$ is large) or harder to learn (their E_m remains large, hence they are chosen again and again).

5. APPLICATION

In this section, we describe the details of the application of hints to forecasting in the FX markets [5]. We start by discussing the very noisy nature of financial data that makes this type of application particularly suited for the use of hints.

A financial market can be viewed as a system that takes in a lot of information (fundamentals, news events, rumors, who bought what when, etc.) and produces an output f (say up/down price movement for simplicity). A model, e.g., a neural network, attempts to simulate the market (figure 11), but it takes an input x which is only a small subset of the information. The 'other information' cannot be modeled and plays the role of noise as far as x is concerned. The network cannot determine the target output f based on x alone, so it approximates it with its output g. It is typical that this approximation will be correct only slightly more than half the time.

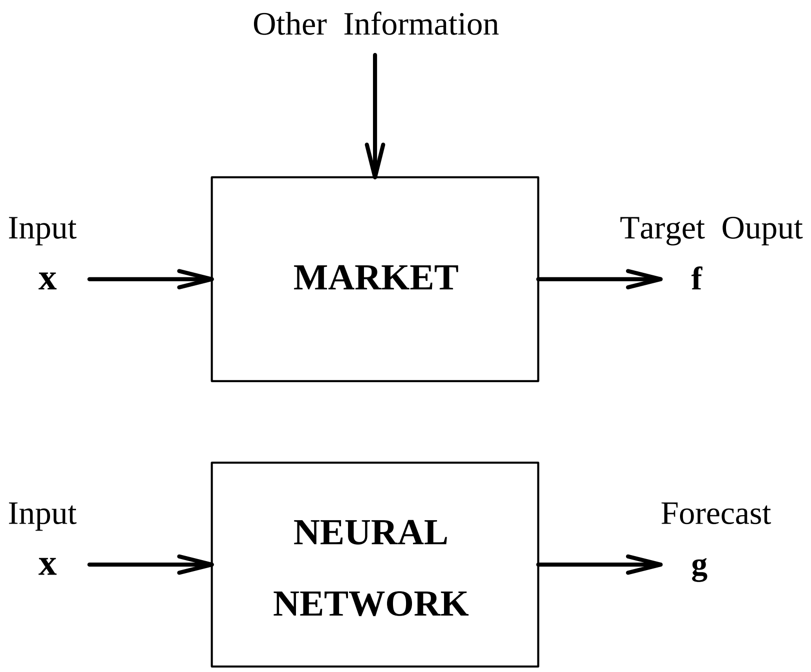

Figure 11: Illustration of the nature of noise in financial markets

What makes us consider x 'very noisy' is that g and f agree only $\frac{1}{2} + \epsilon$ of the time (50% performance range). This is in contrast to the typical pattern recognition application, such as optical character recognition, where g and f agree $1 - \epsilon$ of the time (100% performance range). It is not the poor performance *per se* that poses a problem in the 50% range, but rather the additional difficulty of learning in this range. Here is why.

During learning, we use a limited set of N examples to train (and validate) a neural network to forecast the market. In the 50% performance range, we want g to agree with f on $(\frac{1}{2} + \epsilon)N$ examples. The number of hypotheses on N points that do that is huge. Too many random hypotheses look like good candidates based on the limited set of examples. This is in contrast to the 100% performance range where the hypotheses need to agree with f on $(1 - \epsilon)N$ examples. The number of hypotheses that do that is limited. Therefore, one can have much more confidence in a hypothesis that was learned in the 100% range than one learned in the 50% range. It is not uncommon to see a random trading policy making good money for a few months, but it is very unlikely that a random character recognition system will read a paragraph correctly.

Of course this problem would diminish if we used a very large set of examples, because the law of large numbers would make it less and less likely that g and f can agree $\frac{1}{2} + \epsilon$ of the time just by 'coincidence'. However, financial data has the other problem of nonstationarity. Because of the continuous evolution in the markets, old data may represent patterns of behavior that no longer hold. Thus, the relevant data for training purposes is limited to fairly recent times. Put together, noise and nonstationarity mean that the training data will not contain enough information for the network to learn the function. More information is needed, and hints can be the means of providing it.

Even simple hints can result in significant improvement in the learning performance. Figure 1 showed the learning performance for FX trading with and without the symmetry hint. Figure 12 illustrates this hint as it applies to the U.S. Dollar versus the German Mark. The hint asserts that if a pattern in the price history implies a certain move in the market, then this implication holds whether you are looking at the market from the U.S. Dollar viewpoint or the German Mark viewpoint. Formally, in terms of normalized prices, the hint translates to invariance under inversion of these prices. Notice that the hint says nothing about whether the market should go up or down. It only requires that the prediction be consistent from both sides of this symmetric market.

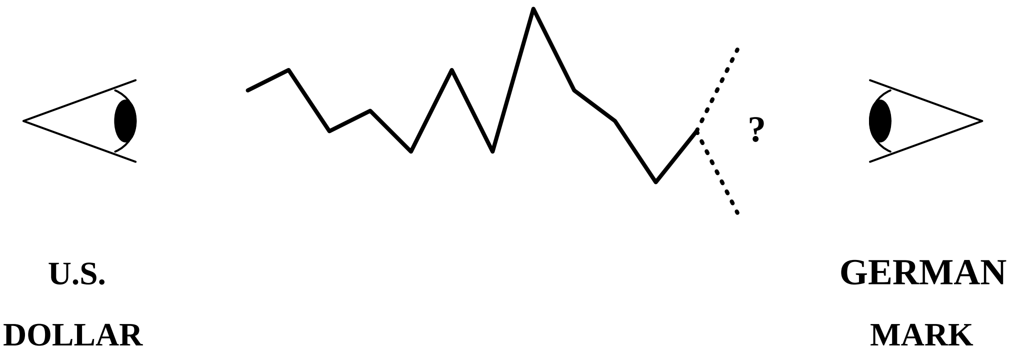

Figure 12: Illustration of the symmetry hint in FX markets

Is the symmetry hint valid? The ultimate test for this is how the learning performance is affected by the introduction of the hint. The formulation of hints is an art. We use our experience, common sense, and analysis of the market to come up with a list of what we believe to be valid properties of this market. We then represent these hints by virtual examples, and proceed to incorporate them in the objective function. The improvement in performance will only be as good as the hints we put in. The idea of soft hints allows us to use hints that are less reliable taking into consideration how much confidence we have in them.

The two curves in figure 1 show the Annualized Percentage Returns (cumulative daily, unleveraged, transaction cost included), for a sliding one-year test window in the period from April 1988 to November 1990, averaged over the four major FX markets with more than 150 runs per currency. The error bar in the upper left corner is 3 standard deviations long (based on 253 trading days, assuming independence between different runs). The plots establish a statistically significant differential in performance due to the use of hints. This differential holds to varying degrees for the four currencies; the British Pound, the German Mark, the Japanese Yen, and the Swiss Franc (versus the U.S. Dollar), as seen in figures 13-16.

In each market, only the closing prices for the preceding 21 days were used for inputs. The objective function we chose was based on the maximization of the total return on the training set, not the minimization of the mean square error, and we used simple filtering methods on the inputs and outputs of the networks. In each run, the training set consisted of 500 days, and the test was done on the following 253

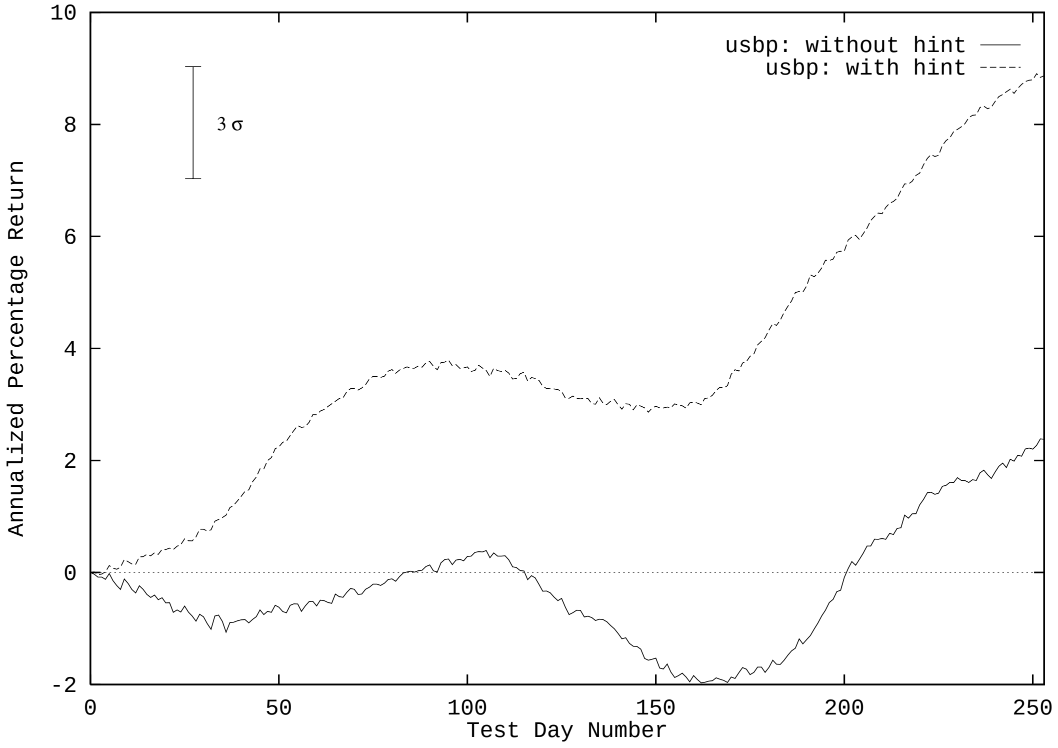

Figure 13: British Pound Performance with and without hint

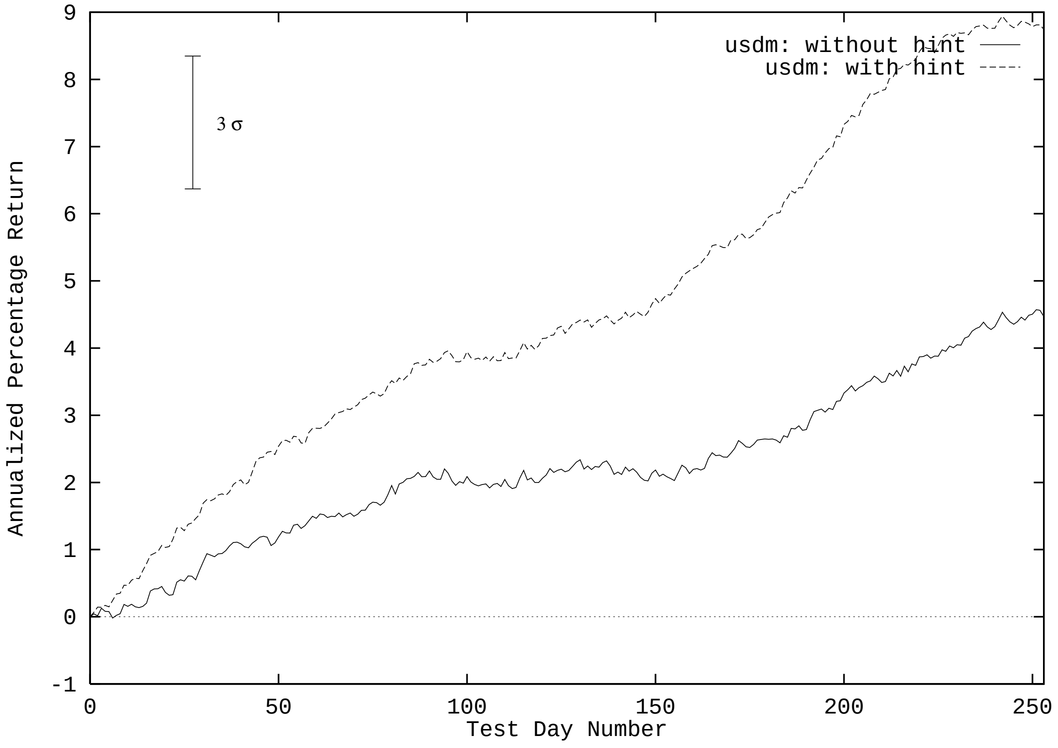

Figure 14: German Mark Performance with and without hint

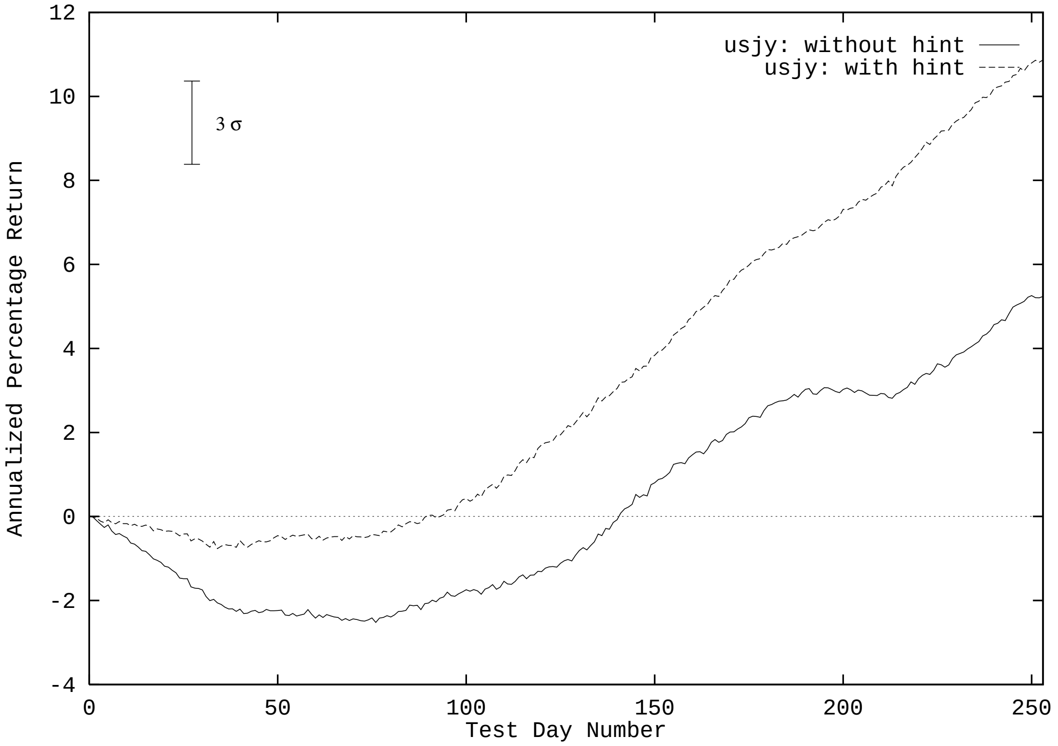

Figure 15: Japanese Yen Performance with and without hint

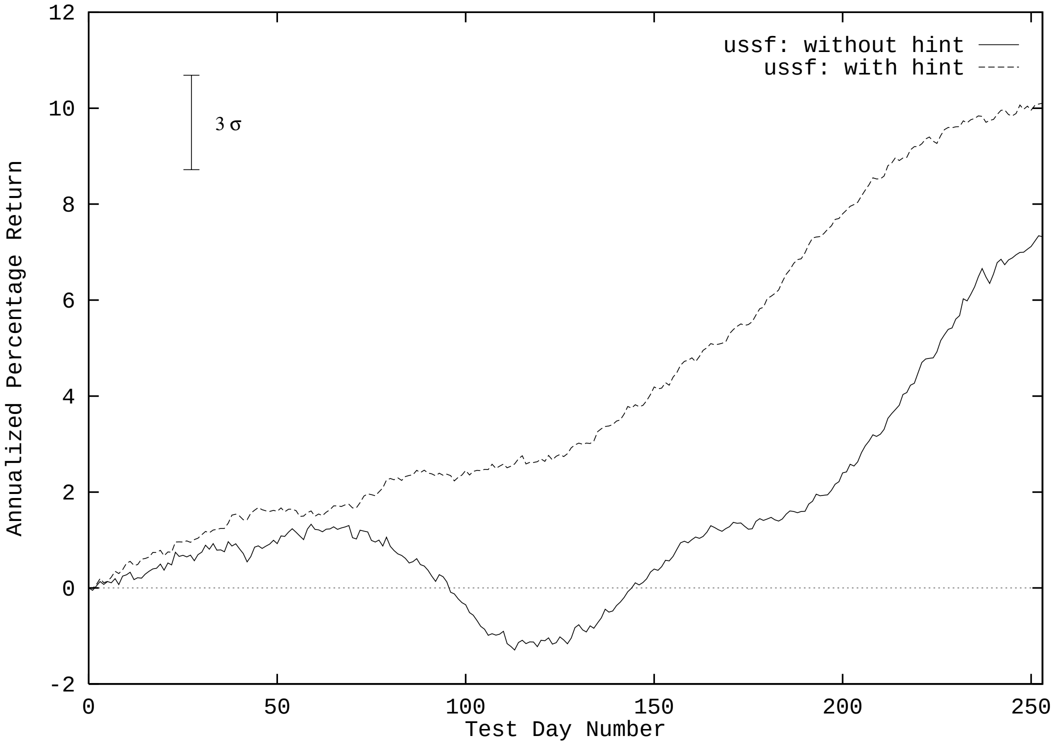

Figure 16: Swiss Franc Performance with and without hint

days. Figures 13-16 show the results of these tests averaged over all the runs. All four currencies show an improved performance when the symmetry hint is used. The statistics of resulting trades are as follows. We are in the market about half the time, each trade takes 4 days on the average, the hit rate (percentage of winning days) is close to 50%, and the Annualized Percentage Return without the hint is about 5% and with the hint is about 10%. Notice that having the return as the objective function resulted in a fairly good return with a modest hit rate.

Since the goal of hints is to add information to the training data, the differential in performance is likely to be less dramatic if we start out with more informative training data. Similarly, an additional hint may not have a pronounced effect if we have already used a few hints in the same application. There is a saturation in performance in any market that reflects how well the future can be forecast from the past. (Believers in the Efficient Market Hypothesis [21] consider this saturation to be at zero performance). Hints will not make us forecast a market better than whatever that saturation level may be. They will, however, enable learning from examples to approach that level.

SUMMARY

The main practical hurdle that faced learning from hints was the fact that hints came in different shapes and forms and could not be easily integrated into the standard learning paradigms. Since the introduction of systematic methods for learning from hints five years ago, hints have become a regular value-added tool. This paper reviewed the method for using different hints as part of learning from examples. The method does not restrict the learning model, the descent technique, or the use of regularization.

In this method, all hints are treated on equal footing, including the examples of the target function. Hints are represented in a canonical way using virtual examples. The performance of the hints is captured by the error measures $E_0, E_1, \cdots, E_M$, and the learning algorithm attempts to simultaneously minimize these quantities. This gives rise to the idea of balancing between the different hints in the objective function. The Adaptive Minimization algorithm achieves this balance by relating the E_m's to the test error E.

Hints are particularly useful in applications where the information content of the training data is limited. Financial applications are a case in point because of the nonstationarity and the high level of noise in the data. We reviewed the application of hints to forecasting in the four major foreign-exchange markets. The application illustrates how even a simple hint can have a decisive impact on the performance of a real-life system.

REFERENCES

[1] Y. Abu-Mostafa, "The Vapnik-Chervonenkis dimension: Information versus Complexity in learning," *Neural Computation* **1** pp. 312-317, 1989.

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

[3] Y. Abu-Mostafa, "Hints and the VC dimension," *Neural Computation* **5**, pp. 278-288, 1993.

[4] Y. Abu-Mostafa, "A method for learning from hints," in *Advances in Neural Information Processing Systems* **5**, pp. 73-80, 1993.

[5] Y. Abu-Mostafa, "Financial market applications of learning from hints," in *Neural Networks in the Capital Markets*, A. Refenes (ed.), Wiley, England, 1994.

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

[7] K. Al-Mashouq and I. Reed, "Including hints in training neural networks," *Neural Computation* **3**, pp. 418-427, 1991.

[8] E. Amaldi, "On the complexity of training perceptrons,,, in *Proceedings of the 1991 International Conference on Artificial Neural Networks (ICANN '91)*, T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas (eds), pp. 55-60, North Holland, 1991.

[9] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, "Learnability and the Vapnik-Chervonenkis dimension," *Journal of the ACM* **36**, pp. 929-965, 1989.

[10] Z. Cataltepe and Y. Abu-Mostafa, "Estimating learning performance using hints," *Proceedings of the 1993 Connectionist Models Summer School*, M. Mozer et al (eds), pp. 380-386, Erlbaum, 1994.

[11] T. Cover and J. Thomas, *Elements of Information Theory*, Wiley-Interscience, 1991.

[12] R. Duda and P. Hart, *Pattern Classification and Scene Analysis,* John Wiley &

Sons, 1973.

[13] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 2, Wiley, 1968.

[14] W. Fyfe, *Invariance hints and the VC dimension,* PhD Thesis, Computer Science Department, Caltech, (Caltech-CS-TR-92-20), 1992.

[15] R. Hecht-Nielsen, *Neurocomputing,* Addison-Wesley, 1990.

[16] G. Hinton, "Learning translation invariant recognition in a massively parallel network," *Proc. Conf. Parallel Architectures and Languages Europe*, pp. 1-13, 1987.

[17] G. Hinton, C. Williams, and M. Revow, "Adaptive elastic models for hand-printed character recognition," in *Advances in Neural Information Processing Systems* **4**, J. Moody, S. Hanson, and R. Lippmann (eds), pp. 512-519, Morgan Kaufmann, 1992.

[18] K. Hertz, A. Krough, and R. Palmer, *Introduction to the Theory of Neural Computation,* Lecture Notes, vol.1, Santa Fe Institute Studies in The Sciences of Complexity, 1991.

[19] M. Hu, "Visual pattern recognition by moment invariants," *IRE Trans. on Information Theory*, vol. IT-8, pp. 179-187, 1962.

[20] J. Judd, *Neural network design and the complexity of learning*, MIT Press, 1990.

[21] B. Malkiel, *A Random Walk Down Wall Street*, W. W. Norton & Co., New York, 1973.

[22] J. McClelland and D. Rumelhart, *Explorations in parallel distributed processing*, MIT Press, 1988.

[23] M. Minsky and S. Papert, *Perceptrons*, expanded edition, MIT Press, 1988.

[24] J. Moody, "The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems," in *Advances in Neural Information Processing Systems* **4**, J. Moody, S. Hanson, and R. Lippmann (eds), pp. 847-854, Morgan Kaufmann, 1992.

[25] J. Moody and J. Utans, "Principled architecture selection for neural networks: Application to corporate bond rating prediction," in *Advances in Neural Information Processing Systems* **4**, J. Moody, S. Hanson, and R. Lippmann (eds), pp. 683-690, Morgan Kaufmann, 1992.

[26] P. Moscato, "An introduction to population approaches for optimization and hierarchical objective functions: A discussion on the role of Tabu Search," *Annals of Operations Research* **41** pp. 85-121, 1993.

[27] C. Omlin and C. Giles, "Training second-order recurrent neural networks using hints," *Machine Learning: Proceedings of the Ninth International Conference* **ML-92**, D. Sleeman and P. Edwards (ed.), Morgan Kaufmann, 1992.

[28] T. Poggio and T. Vetter, "Recognition and structure from one 2D model view: Observations on prototypes, object classes and symmetries," *AI Memo No. 1347*, Massachusetts Institute of Technology, 1992.

[29] D. Rumelhart, G. Hinton, R. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing* **1**, D. Rumelhart et al (eds), MIT Press, pp. 318-362, 1986.

[30] S. Suddarth and A. Holden, "Symbolic neural systems and the use of hints for developing complex systems," *International Journal of Machine Studies* **35**, p. 291, 1991.

[31] V. Vapnik and A. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory of Probability and Its Applications* **16**, pp. 264-280, 1971.

[32] A. Weigend, B. Huberman and D. Rumelhart, "Predicting the future: a connectionist approach," *International Journal of Neural Systems* **1**, pp. 193-209, 1990.

[33] A. Weigend, D. Rumelhart, and B. Huberman "Generalization by weight elimination with application to forecasting," in *Advances in Neural Information Processing Systems* **3**, R. Lippmann, J. Moody, and D. Touretzky (eds), pp. 875-882, Morgan Kaufmann, 1991.

[34] A. Weigend and D. Rumellhart, "Generalization through minimal networks with application to forecasting," in *Proceedings INTERFACE'91–Computing Science and Statistics (23rd Symposium),* E. Keramidas (ed.), Interface Foundation of North America, pp. 362-370, 1991.

[35] D. Wismer and R. Chattergy, *Introduction to Nonlinear Optimization*, North Holland, 1978.