STACKED GENERALIZATION

by David H. Wolpert

Complex Systems Group, Theoretical Division, and Center for Non-linear Studies, MS B213, LANL, Los Alamos, NM, 87545 (dhw@tweety.lanl.gov) (505) 665-3707.

This work was performed under the auspices of the Department of Energy.

LA-UR-90-3460

Abstract: This paper introduces stacked generalization, a scheme for minimizing the generalization error rate of one or more generalizers. Stacked generalization works by deducing the biases of the generalizer(s) with respect to a provided learning set. This deduction proceeds by generalizing in a second space whose inputs are (for example) the guesses of the original generalizers when taught with part of the learning set and trying to guess the rest of it, and whose output is (for example) the correct guess. When used with multiple generalizers, stacked generalization can be seen as a more sophisticated version of cross-validation, exploiting a strategy more sophisticated than cross-validation's crude winner-takes-all for combining the individual generalizers. When used with a single generalizer, stacked generalization is a scheme for estimating (and then correcting for) the error of a generalizer which has been trained on a particular learning set and then asked a particular question. After introducing stacked generalization and justifying its use, this paper presents two numerical experiments. The first demonstrates how stacked generalization improves upon a set of separate generalizers for the NETtalk task of translating text to phonemes. The second demonstrates how stacked generalization improves the performance of a single surface-fitter. With the other experimental evidence in the literature, the usual arguments supporting cross-validation, and the abstract justifications presented in this paper, the conclusion is that for almost any real-world generalization problem one should use some version of stacked generalization to minimize the generalization error rate. This paper ends by discussing some of the variations of stacked generalization, and how it touches on other fields like chaos theory.

Key Words:

generalization and induction, combining generalizers, learning set pre-processing, cross-validation, error estimation and correction.

INTRODUCTION

This paper concerns the problem of inferring a function from a subset of $\mathbf{R}^n$ to a subset of $\mathbf{R}^p$ (the *parent* function) given a set of m samples of that function (the *learning set*). The subset of $\mathbf{R}^n$ is the *input space*, and the subset of $\mathbf{R}^p$ is the *output space*. A *question* is an input space (vector) value. An algorithm which guesses a parent function, basing the guess only on a learning set of m $\mathbf{R}^{n+p}$ vectors read off of that parent function, is called a *generalizer*. A generalizer guesses an appropriate output for a question via the parent function it infers from the learning set. For simplicity, although the analysis of this paper holds for any positive integer p, unless explicitly indicated otherwise I will always take p = 1.

In this paper I am usually assuming noiseless data. This means that the best guesses for the inputs of elements of the learning set are already known to us - they are provided by the learning set. Building a system to guess properly for those elements - i.e., "learning that learning set" - is trivial, and can be achieved simply by building a look-up table. (Difficulties only arise when one insists that the look-up table be implemented in an odd way, e.g., as a feedforward neural net.) Therefore in this paper the questions of interest are almost always outside of the learning set.

Some examples of generalizers are back-propagated neural nets (Rumelhart and McClelland 1986), Holland's classifier system (Holland 1975), and Rissanen's minimum description length principle (Rissanen 1986) (which, along with all other schemes which attempt to exploit Occam's razor, is analyzed in (Wolpert 1990a)). Other important examples are memory-based reasoning schemes (Stanfill and Waltz 1986), regularization theory (Poggio et al. 1988), and similar schemes for overt surface fitting of a parent function to the learning set (Wolpert 1989, Wolpert 1990a, Wolpert 1990b, Farmer and Sidorowich 1988, Omohundro 1987).

In this paper I will primarily be interested in generalizers which are capable of guessing as output a number which does not occur as an output value in the learning set. Conventional classifiers (e.g. ID3 (Quinlan 1986), Bayesian classifiers like Schlimmer's Stagger system (Dietterich

1990), etc.) don't have this flexibility, although in all other respects they are valid examples of generalizers.

This paper introduces stacked generalization, a technique whose purpose is to achieve a generalization accuracy (as opposed to learning accuracy) which is as high as possible. The central idea is that one can do better than simply list all guesses as to the parent function which are consistent with a learning set (as is done in PAC-style learning (Dietterich 1990, Valiant 1984), for example). One can also use in-sample/out-of-sample techniques to try to find a best guesser of parent functions (or to try to find a best combination of guessers of parent functions). By creating a partition of the learning set, training on one part of the partition, and then observing behavior on the other part, one can try to deduce (and correct for) the biases of one or more generalizers with respect to that learning set. Loosely speaking, in addition to finding all theories consistent with a set of data, by means of partitions of the data one can also construct a best theorist, and then use whichever theory it prefers.[1]

There are many different ways to implement stacked generalization. Its primary implementation is as a technique for combining generalizer, although it can also be used when one has only a single generalizer, as a technique to improve that single generalizer.

For any real-world learning set θ, there are always many possible generalizers $\{G_j\}$ one can use to extrapolate from θ. One is *always* implicitly presented with the problem of how to address this multiplicity of possible generalizers. Most algorithmic schemes for addressing this problem, including in particular non-parametric statistics techniques like cross-validation (Efron 1979, Stone 1977), generalized cross-validation (Li 1985) and bootstrapping (Efron 1979), are winner-takes-all strategies. These schemes can be viewed as mappings which take an arbitrary generalizer and learning set as input, and give as output an estimate of the average generalizing accuracy of that generalizer, for the unknown parent function which generated the learning set. To use such a mapping one simply picks that $G \in \{G_j\}$ which, together with θ, has the highest estimated generalization accuracy according to the mapping, and then uses that G to generalize from θ.

In contrast, stacked generalization provides a strategy for this situation which is more sophisticated than winner-takes-all. Loosely speaking, this strategy is to combine the $\{G_j\}$ rather than choose one amongst them. This can be done (for example) by taking their output guesses as input components of points in a new space, and then generalizing in that new space. (See figure 1.)

Later on in this paper winner-takes-all strategies will be shown to be a special case of using stacked generalization in this manner, where one is doing the generalization in the "new space" by means of a global fit of a highly restricted hyperplane. Accordingly, stacked generalization can be viewed as a more sophisticated version of non-parametric statistics techniques like cross-validation. In particular, all the usual arguments supporting such techniques apply even more strongly to stacked generalization, and therefore it can be argued that for almost *any* generalization or classification problem, since invariably there is more than one generalizer which can be applied to the problem, to maximize the generalization accuracy one should use stacked generalization rather than any single generalizer by itself.[2]

In addition to viewing it as an extension of concepts like cross-validation, stacked generalization can also be viewed as a means of collectively using all of the $\{G_j\}$ to estimate their own generalizing biases with respect to a particular learning set, and then filter out those biases. This description is particularly apt in the variation of stacked generalization appropriate when one only has a single generalizer. In such a situation, stacked generalization is (overtly) a scheme for estimating the errors of a generalizer when working on a particular learning set, and then correcting those errors. (See figure 2.)

Section I of this paper presents a rigorous definition of stacked generalization and discusses why it would be expected to improve generalization accuracy. Section II of this paper then presents two experimental examples of using stacked generalization. The first is using it to improve the performance of a single generalizer (here an explicit surface-fitting algorithm). The second is using it to improve upon the individual performance of several generalizers for a modified version of the text-to-phoneme data set that went into making NETtalk (Stanfill and Waltz 1986, Wolpert 1990b, Carterette and Jones 1974, Sejnowski and Rosenberg 1988). Section III then discusses some of the myriad variations and extensions of stacked generalization, some of the ways it can be approached theoretically, and some heuristics concerning the scheme's behavior.

It would be impossible to investigate in great depth all of the theoretical and empirical issues concerning stacked generalization in a single paper. This paper is instead intended to serve as a broad introduction to the idea of stacked generalization and its many variations.

I. HOW STACKED GENERALIZATION WORKS

For full rigor, I will first provide a mathematical definition of a generalizer. Then I will define the process of stacked generalization, giving a rigorous definition of cross-validation along the way. Unfortunately, it's in the nature of stacked generalization that presenting it in full generality and full rigor makes it appear more complicated than it really is. At the end of this section, two examples are provided to mitigate this effect. The first example is of a way to use stacked generalization with multiple generalizers, and the second example is of a way to use it with a single generalizer.

i) Generalizers

A generalizer is a mapping taking {a learning set of m pairs $\{x_k \in \mathbf{R}^n, y_k \in \mathbf{R}\}$, $1 \le k \le m$, together with a question $\in \mathbf{R}^n$} into {a guess $\in \mathbf{R}$}. (Full generality would have the guess $\in \mathbf{R}^m$, not $\mathbf{R}$. However for most applications one can replace a generalizer making guesses in $\mathbf{R}^m$ with the Cartesian product of m separate generalizers making guesses in $\mathbf{R}$. Accordingly, in this paper m is taken to equal 1. See (Wolpert 1989).) For a given n, such a mapping is equivalent to a countably infinite set of functions $\{g_i\}$, $1 \le i < \infty$, one function for each possible value of m. g_1 takes three arguments (the learning set input x_1, the learning set output y_1, and the question q); g_2 takes five arguments (x_1, y_1, x_2, y_2, and q); and so on (see (Wolpert 1990c)). Often the $\{g_i\}$ are only implicitly defined in the definition of the generalizer's algorithm. This is the case with back-propagation, for example. (Strictly speaking, the $\{g_i\}$ of back-propagation aren't single-valued, since they depend on the (random) initial choice of weights. This difficulty is of little consequence however, and can be avoided explicitly by averaging over a set of initial choices of the weights, for example.) In other generalizers (e.g. generalizers which work by explicitly fitting a surface), it's possible to write down the $\{g_i\}$ directly. Colloquially, one says that a generalizer's g_m, when provided with the argument list $\{x_1, y_1, x_2, y_2, ..., x_m, y_m; q\}$, is being "taught" or "trained" with an m-element learning set consisting of the elements $\{x_1, y_1, x_2, y_2, ..., x_m, y_m\}$, and is then "asked" the question q, for which it "guesses" what the corresponding output should be. If the generalizer returns the appropriate y_i whenever q is equal to one of the x_i in the learning set, then we say that the generalizer *reproduces* the learning set.

In the scenario considered by this paper, we are given a learning set θ of m elements living in the space $\mathbf{R}^{n+1}$. Together with θ we are given a set of N generalizers $\{G_j\}$, where $N \geq 1$ (i.e., we're given a set of N separate sequences of functions $\{g_i\}$). As an example, n could be 3, and the learning set might consist of m elements of the form (a, b, c, output = a + b + c), where a, b, and c are integers. "Correct" generalization would be to guess the parent function {output = sum of the three input components}. N could then be four, for example, with the four G_j being ID3, back-propagation, global fitting with a polynomial (over the n variables) of minimal order, and a local surface-fitting technique. Since it is a classifier (see introduction), we know that ID3 can not generalize correctly for this parent function unless it is attached to a de-trending pre-processor. Similar difficulties will affect back-propagation (see (Wolpert 1989)). Of course, none of this can be known with certainty to someone only provided with the learning set θ and not with the entire parent function.

In what follows I will often be a bit free with the symbols, and write G(θ; q), for example, when what I really mean is the output of the generalizer G's m'th function, g_m (where m is the number of elements in the provided learning set θ), taking as argument list the enumerated elements of θ followed by the question q. Similarly, even though it itself is made up of components (being an n-dimensional vector), I will often refer to the input space projection of a point in the full input/output space $\mathbf{R}^{n+1}$ as the "input component" of that point. Moreover, I will often refer to the "nearest neighbor" of a point in a space. What I really mean is the nearest neighbor of that point as measured in the input space projection of the full space. For these and all other cases, the context should always make the meaning clear.

ii) Partition sets and cross-validation

The first step in employing stacked generalization is choosing a set of r partitions, each of which splits θ into two (usually disjoint) sets. Label such a set of partitions as θ_{ij}, where $1 \leq i \leq r$, and $j \in \{1, 2\}$. Such a set of partitions is called a *partition set*. For example, for a cross-validation partition set (CVPS), r = m, for all i θ_{i2} consists of a single element of θ, the corresponding θ_{i1} consists of the rest of θ, and $\theta_{i2} \neq \theta_{j2}$ for $i \neq j$. (Since r = m, this last requirement of distinctness of the θ_{i2} means that the set of all θ_{i2} covers θ). One pair of such a CVPS is illustrated in both figures 1 and figure 2. One can define a bootstrap partition set in a similar way to a CVPS, except that (roughly speaking) the elements θ_{i2} are chosen randomly rather than in such a way as to exhaustively cover θ with no duplications. As another example, for a GMDH partition set (Xiangdong and Zhaoxuan 1990) r is some divisor of m, θ_{i1} consists of m/r elements, $\theta_{i1} \cap \theta_{j1} = \emptyset$ for $i \neq j$, and as with the CVPS $\theta_{i2} = \theta - \theta_{i1}\ \forall$ i. (Here it's the θ_{i1} which form a disjoint cover for θ rather than the θ_{i2}.) Partition sets of this type where r < m are particularly useful if it takes a long time to train some of the $\{G_j\}$.

The winner-takes-all technique of cross-validation is a very straight-forward way to use a CVPS to map a generalizer G together with a learning set θ to an estimate of the generalization error rate of G when generalizing from θ. Intuitively, it estimates the generalization accuracy by seeing how well the generalizer can guess one part of the full learning set when taught with the rest of it. More rigorously, it works with the CVPS by calculating the average, over all i, of the error of G at guessing what output corresponds to the input component of θ_{i2} when taught only with the remainder of θ, θ_{i1}: the cross-validation error estimate of G with respect to θ is defined by

$$\text{C.V.}(G, \theta) \equiv \Sigma_i\, [G(\theta_{i1}; \text{the input component of } \theta_{i2}) \ - \ (\text{the output component of } \theta_{i2})]^2 \, / \ m.$$

The technique of minimal cross-validation error says that given a set of candidate generalizers $\{G_j\}$ and a learning set θ, one should generalize from θ with that generalizer $G_k \in \{G_j\}$ such that $\text{C.V.}(G_k, \theta) < \text{C.V.}(G_j, \theta) \ \forall\, j \neq k$.

For simplicity, in the rest of this paper, we will only consider the CVPS, so any set θ_{i2} consists of only one element.

iii) Stacked generalization

Define the $\mathbf{R}^{n+1}$ space inhabited by the original learning set θ as the "level 0 space". Any generalizer when generalizing directly off of θ in the level 0 space is called a "level 0" generalizer, and the original learning set θ is called a "level 0" learning set. For each of the r partitions of θ, $\{\theta_{i1}, \theta_{i2}\}$, look at a set of k numbers determined by (a subset of) the N $\{G_j\}$ working together with that partition. Typically these k numbers can be things like the guesses made by the $\{G_j\}$ when taught with θ_{i1} and presented as a question the input component of the element θ_{i2} (i.e., $G_j(\theta_{i1}$; the input component of θ_{i2})), the input component of the element θ_{i2}, or the vector in the input space connecting that input component of θ_{i2} to its nearest neighbor in θ_{i1}. Take each such set of k numbers and view it as the input component of a point in a space $\mathbf{R}^{k+1}$. The corresponding output value of each such point is calculated from the output component of the corresponding θ_{i2}, perhaps along with $G_j(\theta_{i1}$; the input component of $\theta_{i2})$ for one of the $\{G_j\}$. This space $\mathbf{R}^{k+1}$ is called the "level 1 space". Since we have r partitions of θ, we have r points in the level 1 space. Those r points are known as the "reduced" or the "level 1" learning set. (In figures 1 and 2, the level 1 learning set is L'.)

We wish to generalize from θ by operating a generalizer in the level 1 space. We can do this in many ways. The common idea is to take a question in the level 0 space, pass it through the transformations which produced the input components of the level 1 learning set to get a level 1 question in the level 1 input space, and then answer that level 1 question by generalizing from the level 1 learning set. This level 1 guess is then transformed back into a level 0 guess. (Said transformation being determined by how the output components of the θ_{i2} are used to calculate the output components of the level 1 learning set.) Any generalizing process of this form is known as "stacked generalization". The process as a whole can be iterated, resulting in levels p > 1 (i.e., multiple stackings). For now, we'll just be concerned with 2 levels stacked generalization, as described above.

It is important to note that many aspects of stacked generalization are, at present, "black art". For example, there are currently no hard and fast rules saying what level 0 generalizers one should use, what level 1 generalizer one should use, what k numbers to use to form the level 1 input space, etc. In practice, one must usually be content to rely on prior knowledge to make (hopefully) intelligent guesses for how to set these details. Of course, the same use of "black art" occurs in the rest of machine learning as well. For example, in practice most researchers currently rely on prior knowledge of the problem domain to make a (hopefully) intelligent guess as to what generalizer to use and how to configure it.

iv) An example of stacked generalization for the case of multiple generalizers

As an example of stacked generalization, assume we have an m-element learning set θ of points living in $\mathbf{R}^{n+1}$, a set of N generalizers $\{G_j\}$, and a question $q \in \mathbf{R}^n$. As was mentioned before, we're restricting ourselves to the CVPS. This partition set gives us m sets $\{\theta_{i1}, \theta_{i2}\}$, where each θ_{i1} is a different subset of m - 1 of the elements of θ, and θ_{i2} is the remaining element of θ. Let k = N, and have the k = N numbers used to construct the input components of an element of the level 1 learning set be the guesses made by all N of the $\{G_j\}$ when taught with a particular θ_{i1} and presented with the input component of the corresponding θ_{i2} as a question. (In other words, a particular point in the level 1 learning set has the N components of its input projection set to the N numbers $G_j(\theta_{i1}$; the input component of $\theta_{i2})$. See figures 1, 3, and 4.) Let the output component of a point in the level 1 learning set be given directly by the output component of the corresponding θ_{i2}. Since there are r = m partitions of θ, there are r = m elements in the level 1 learning set, just like in the level 0 learning set. Since k = N, each point in this level 1 learning set has an N-dimensional input component. To make a guess for the level 0 question q, we convert it into a level 1 question. We do this in the same way we made the level 1 learning set: we find the guess made in response to the question q by all N of the $\{G_j\}$ when taught with (now the full) learning set θ. These N guesses give us the input coordinates of a question in the level 1 space, and to answer *that* question we simply run some generalizer off of the level 1 learning set and ask it that level 1 question. This guess for what level 1 output should correspond to the level 1 question is then taken as the guess made by the entire stacked generalization process for what level 0 output should correspond to the original level 0 question.

This procedure sounds complicated. It really isn't. As an example, take the parent function "output = sum of the three input components" mentioned in section I(i). Our learning set θ might consist of the five input-output pairs (0, 0, 0; 0), (1, 0, 0; 1), (1, 2, 0; 3), (1, 1, 1; 3), (1, -2, 4; 3), all sampled with no noise from the parent function. Label these five input-output pairs as θ_{12} through θ_{52}, with $\theta_{i1} \equiv \theta - \theta_{i2}$ (so for example θ_{21} consists of the four pairs {(0, 0, 0; 0), (1, 2, 0; 3), (1, 1, 1; 3), (1, -2, 4; 3)}. We have two level 0 generalizers, G_1 and G_2, and a single level 1 generalizer, Γ. The level 1 learning set L' is given by the five input-output pairs ($G_1(\theta_{i1}$; input components of θ_{i2}), $G_2(\theta_{i1}$; input components of θ_{i2}); output component of θ_{i2}) given by the five possible values of i. (This level 1 space has two dimensions of input and one of output.) So for example the member of the level 1 learning set corresponding to i = 1 has output component 0 and input component $G_1(\theta_{11}; (0, 0, 0))$, $G_2(\theta_{11}; (0, 0, 0))$. Now we are given a level 0 question (x_1, x_2, x_3). We answer it with the guess $\Gamma(L' ; (G_1(\theta; (x_1, x_2, x_3)), G_2(\theta; i(x_1, x_2, x_3))))$, i.e., we answer it by training Γ on L' and then asking it the question given by the guesses of the two level 0 generalizers which were themselves trained on all of θ and asked the question q.

The guess made by this implementation of stacked generalization is determined by combining the guesses of the original N $\{G_j\}$. *How* they are combined depends on the level 1 generalizer used. For example, consider the following level 1 generalizer: "Fit the (level 1) learning set with a single global hyperplane of the form {output = value of input dimension t}. There are k such global hyperplane fits for the k possible values of t; choose the hyperplane fit with the smallest RMS Euclidean error for fitting the level 1 learning set." In the language of pattern recognition, this generalizer is the rule "find which single feature (i.e., which input space component) has the greatest correlation with the correct answer, and guess according to it". This level 1 generalizer results in a winner-takes-all strategy for using the $\{G_j\}$. It makes the determination of which of the $\{G_j\}$ to use by finding the G_j with the minimal RMS error for predicting part of the level 0 learning set from the rest. This error is calculated using the cross-validation partition set. In fact, a moment's thought shows that stacked generalization with this simple-minded level 1 generalizer is the exact same generalizing process as the technique of minimal cross-validation! As was mentioned in the introduction, cross-validation is seen to be just a (relatively uninteresting) special case of stacked generalization, corresponding to an extraordinarily dumb level 1 generalizer.[3]

As another naive example, the level 1 generalizer could be independent of the level 1 learning set: "make a guess for a (level 1) question by averaging all the k components of that question". Another extraordinarily dumb level 1 generalizer. Yet the guesses it makes are the same as those made by perhaps the most common currently used scheme (second to winner-takes-all schemes) for combining generalizers; stacked generalization with this level 1 generalizer is exactly equivalent to simply averaging the guesses made by all N of the $\{G_j\}$. A (marginally) more sophisticated way of combining generalizers is to form a weighted average of the guesses of the $\{G_j\}$. This is equivalent to having the level 1 generalizer be a scheme to fit the level 1 learning set with a single global hyperplane.

We can view the various commonly used schemes for combining generalizers as simply special cases of stacked generalization. In all these schemes, one is implicitly confronted with a level 1 learning set and must decide how to generalize from it. Yet the problem of how to generalize from the level 1 learning set is just a normal generalization problem, in principle no different from any other. Therefore, just as with any other generalization problem, it makes no sense to use "dumb" generalizers to generalize from the level 1 learning set. Yet one very noticeable feature of these commonly used schemes for combining (level 0) generalizers is precisely the lack of sophistication of their level 1 generalizers. Therefore, just as with any other generalization problem, one would expect improved performance - perhaps extremely improved performance - if these dumb level 1 generalizers were replaced with more sophisticated generalizers.

v) An example of stacked generalization for the case of one generalizer

There is no a priori reason why the k numbers used to make the level 1 input space have to all be the guesses of a set of generalizers. Nor is there any a priori reason why the output components of the level 1 learning set have to be given directly by the output components of the θ_{i2}. This can be illustrated with an example of how to used stacked generalization when the set $\{G_j\}$ consists of a single element (i.e., by an example of how to use stacked generalization to improve the behavior of a single generalizer, as opposed to using it as a means of combining a set of generalizers).

This example is illustrated in figure 2. (Another similar example is illustrated in figure 5.)

Again use the CVPS, so r = m. The level 0 input space has dimension n; let k = 2n. The 2n numbers defining the level 1 input space are the n coordinates of a level 0 question (like the input components of θ_{i2}) together with the n input coordinates of the vector connecting the nearest neighbor of that question amongst the level 0 learning set (like the nearest neighbor amongst the θ_{i1}) to the question itself. Our single generalizer G doesn't in any way contribute to the level 1 input space values. Rather G comes in, indirectly, in the level 1 space outputs; the output component of a point in the level 1 space is the error (or estimate thereof, as the case might be) of G when trying to guess what output corresponds to the associated level 0 question. For example, in forming the level 1 learning set we set the output value corresponding to a particular partition $\{\theta_{i1}, \theta_{i2}\}$ to be $\{G(\theta_{i1}$; input component of $\theta_{i2})$ - (the output component of $\theta_{i2})\}$. To make a guess as to what output should correspond to the question q, after we've constructed the level 1 learning set we train G with all of θ, ask it q, and store the resultant guess; call it y. Now we feed q, together with the level 0 input space vector connecting q to its nearest neighbor amongst θ, into the level 1 space as level 1 input coordinates. Generalizing in this level 1 space, we get a guess for what level 1 output should correspond to this level 1 question, i.e., we get an estimate for the difference between y and the correct guess. Now subtract half of this error estimate from y to get the number which is our final guess as to what output should correspond to the original question q.

In this procedure we multiply by the constant one half just to be conservative. Note that this multiplicative constant gives us a knob determining how much we're using stacked generalization. When the constant equals 0, our guessing is equivalent to using the level 0 generalizer straight. As the value of this constant is increased, our guessing becomes more and more stacked-generalization-based.

Intuitively, this implementation of stacked generalization with a single generalizer is a means of estimating the actual error (not just the average value of the errors) of the provided generalizer when presented with a particular question and a particular learning set. It works by first seeing how the generalizer errs when taught with only part of the learning set and asked a question in the remainder of the learning set; this information then serves as the level 1 learning set, and the level 1 generalizer generalizes from this information to make an estimate for the error when the original level 0 generalizer is taught with the entire level 0 learning set. This error estimate (or more usually a fraction of it) is then subtracted from the level 0 generalizer's guess to arrive at an improved guess.

The information we send into the level 1 input space determines how our error estimates are allowed to vary. In the example given above, in addition to depending strongly on the level 0 question, we're assuming that the errors of the level 0 generalizer are strongly dependent on the nearest neighbor of that question amongst the elements of the learning set. The rationale is that varying the nearest neighbor of the question often has a pronounced affect on the generalization accuracy of the level 0 generalizer, especially if that level 0 generalizer is something like a local surface-fitter.

It's interesting to note that a special case of single generalizer stacked generalization is exactly equivalent to running the level 1 generalizer by itself. Have the level 1 input values be simply the level 0 question (or input component of θ_{i2}, as the case might be). Furthermore, have the level 1 outputs be the level 0 outputs (i.e., have the transformation taking the output component of the θ_{i2} to the output components of the level 1 space be the identity mapping). Note that no level 0 generalizer is being used. In fact, this entire stacked generalizer structure is exactly equivalent to running the level 1 generalizer by itself directly on the level 0 learning set and level 0 question. Therefore, just as stacked generalization corresponds to an extension of cross-validation when one has multiple generalizers, so when one has only a single generalizer stacked generalization corresponds to an extension of using that generalizer directly by itself.

II. EXPERIMENTAL TESTS OF STACKED GENERALIZATION

This section reports the results of two numerical experiments which indicate that stacked generalization does indeed improve generalization accuracy. These experiments are relatively controlled, "toy" problems. The idea is to use them as pedagogical and heuristic tools, much like the toy problems used in (Rumelhart and McClelland 1986). It should be noted, however, that there appears to be little (if any) degradation of performance of stacked generalization when it is instead applied to messy, real-world problems. For example, Gustafson et al. have reported that (what amounts to) stacked generalization beats back-propagation for some hydrodynamics problems (Gustafson et al. 1990).[4] Similarly, a number of researchers have reported on the efficacy of simply averaging a set of generalizers, for example for aspects of the problem of predicting protein structure (Schulz et al. 1974). He Xiandong has reported on the efficacy of (what amounts to) using stacked generalization with a variant of a GMDH partition set together and a radial basis function generalizer for time-series prediction (Xiangdong and Zhaoxuan 1990).[5] Finally, work in progress with Alan Lapedes and Rob Farber suggests that using stacked generalization to combine ID3, perceptrons, and the author’s metric-based HERBIE (Wolpert 1990b) for the problem of predicting splice junctions in DNA sequences gives accuracy better than any of the techniques by itself (i.e., preliminary evidence indicates that this implementation of stacked generalization is the best known generalization method for this problem).

i) Experiment one

The simpler of the two numerical experiments involved using stacked generalization to improve the performance of a single generalizer. The level 0 input space for this experiment was one-dimensional. The problem was explicitly one of surface-fitting; the parent functions were simple high-school-math-type functions, and the level 0 generalizer was "linearly connect the dots of the learning set to make an input-output surface which then serves as a guess for the parent function", i.e., the local linear technique of Farmer and Sidorowich (1988). (See figure 5 for an illustration of this level 0 generalizer.)

In this experiment the stacked generalization architecture was exactly the same as in the example at the end of section I on how to augment the performance of a single generalizer (see figure 2). n equals 1 for this problem, so the level 1 input space was 2-dimensional. The level 1 generalizer was the metric-based HERBIE described in (Wolpert 1990b, Wolpert 1990c). It works by returning a normalized weighted sum of the outputs of the p nearest neighbors of the question amongst the learning set. Here p was 3, and the weighting factor for each of the 3 nearest neighbors was the reciprocal of the distance between that neighbor and the question. "Normalization" means that the weighted sum was divided by the sum of the weighting factors: guess = $\{ \sum_{i=1}^{3} y_i / d(q, x_i)\} / \{ \sum_{i=1}^{3} 1 / d(q, x_i)\}$, where q is the question, x_1, x_2, and x_3 are the input components of the three nearest neighbors of q in the learning set, y_1, y_2, and y_3 are the corresponding outputs, and d(., .) is a metric, here taken to be the Euclidean metric. (When the input space is symbolic, it is conventional to use a Hamming metric rather than a Euclidean metric. See(Wolpert 1990b).) This metric-based HERBIE is one of the simplest generalizers there are.[6] In addition, along with (for example) Farmer's local linear technique, metric-based HERBIEs necessarily always reproduce their learning set exactly (see (Wolpert 1990b) and (Wolpert 1990c)). The parameters of this use of stacked generalization (e.g., .5, 3) were chosen in an ad hoc manner; presumably cross-validation could be used to get better values.

In the first phase of the experiment stacked generalization was run 1,000 times. Each time a new 3rd order polynomial was created, all four of whose coefficients were chosen randomly from the interval [-2.0, 2.0]. (Here and elsewhere "random" means i.i.d. with a flat sampling distribution.) For each such polynomial parent function a 100-point learning set was chosen randomly, and then a separate 100-point testing set of input space values was chosen randomly. Both sets had their input space values restricted to the interval [-10.0, 10.0]. The learning set was then used to "train" the stacked generalization structure described above, and the errors when using that structure to guess the outputs of the elements of the testing set were recorded and compared to the errors of the level 0 generalizer run by itself with no level-1 post-processing. The average of the difference {(square of the error for the level 0 generalizer run by itself) - (square of the error for the stacked generalizer)} equalled 81.49. The estimated error in this average was +/- 10.34, i.e. stacked generalization improved the generalization with a confidence of 8 standard deviations.

The magnitudes of the guessing errors, both for the level 0 generalizer run straight and for the stacked generalization structure, ranged over many orders of magnitude, so the number "81.49" isn't particularly meaningful. Ratios of error magnitudes can be misleading, but they do have the advantage that (unlike simple differences of error magnitudes) they aren't skewed by such logarithmically broad distributions. The average of the ratio {(square of the error for the level 0 generalizer run by itself) / (square of the error for the stacked generalizer)} equalled 1.929. The estimated error in this average was +/- .0243; using stacked generalization improved the generalization by a factor of 2, on average.

The same problem was first investigated for parent functions which were polynomials only of order 2. The level 1 input space consisted of the two numbers α and β, where $\alpha \equiv (q - x_1)$, $\beta \equiv (q - x_2)$, q is the question, and x_1 and x_2 are the two elements of the learning set used to make the local linear guess (i.e., they're the nearest neighbor of q, and the next nearest neighbor which lives on the opposite side of q from that first nearest neighbor.) For this scenario the average of the ratio of the error magnitudes ranged up to 50 (!), depending on the precise parameters used.

It's not hard to understand this behavior. For polynomials of order two it turns out that the error of the local linear technique is independent of the question. In fact, up to an overall proportionality constant, it's given exactly by $\alpha\beta$. For this scenario, the level 1 generalizer only has to learn the simple surface {output = a constant times the product of the 2 input coordinates}, i.e., a paraboloid, a two-dimensional version of the original parent surface. Let the cardinality of the level 0 learning set be m, and let the range of the input values in that learning set be z. The density in the input space of the elements of the level 0 learning set is ~ m/z. This means that the values of α and β are ~ z/m. Since there are m such values in the level 1 learning set, the density in the input space of the elements of the level 1 learning set ~ $m / (z/m)^2 = (m^3)/(z^2) \sim (m^2)/z$ (under the assumption

$m \cong z$). Since these level 1 points lie on the two-dimensional version of the level 0 parent surface, the stacking of the generalizer has effectively allowed us to run the original generalizer over a learning set chosen from the original surface, but with a density m times that of the original level 0 learning set. We have a "multiplier effect".

As another way of understanding the exceptional behavior for order two polynomial parent surfaces, let the average output space magnitude of the points in the level 0 learning set be s, and let rs be the average error of the level 0 generalizer run straight. r measures the efficacy of the generalizer, and will in general be below 1, fairly close to 0. The average output space magnitude of the points in the level 1 learning set is rs. Since these points lie on the "same" surface as the points of the level 0 learning set, if the same generalizer is used we would expect an average error of the guesses in the level 1 space to be $\sim r \times (rs) = r^2 s << rs$. Just as in the argument of the preceding paragraph, this output space argument says that for polynomials of order two, using a level 1 generalizer with inputs α and β results in a "multiplier effect" diminishing the average guessing error polynomially.

In addition to polynomials, simple transcendental parent functions were also investigated. The level 1 input space was again two-dimensional, the input coordinates were again q and $q - x_1$, and the error estimate made by the level 1 generalizer was again multiplied by .5. Random constants were again chosen from [-2.0, 2.0], the level 0 inputs were again chosen from [-10.0, 10.0], and again 1,000 runs of random 100-point learning sets and random 100-point testing sets were investigated. The same level 0 and level 1 generalizers were used as in the polynomial tests. The parent functions were a sum of two sine waves and two exponential functions. The amplitudes of all four functions were determined by the random constants, as were the phases of the two sine waves (thereby introducing cosines) and the frequencies of the two exponentials. The frequencies of the two sine waves were .1 and .2, and the sine function used interpreted its arguments as being in radians.

The average of the difference {(square of the error for the level 0 generalizer run by itself) - (square of the error for the stacked generalizer)} equalled .0078. The estimated error in this average was +/- .0011, i.e. stacked generalization again improved the generalization with a confidence of 8 standard deviations. The average of the ratio {(square of the error for the level 0 generalizer run by itself) / (square of the error for stacked generalizer)} equalled 2.022. The error in this average was +/- .0318; using stacked generalization again improved the generalization by a factor of 2, on average.

These results are not intended to constitute the definitive investigation of how to use stacked generalization to improve the accuracy of the local linear generalizing technique. Many variations of the schemes outlined here could be investigated (involving, for example, different level 1 generalizers, different values of parameters, different mappings from partitions to a level 1 space, different dimensionalities of the level 1 input space, etc.) Rather these results are simply intended to indicate that stacked generalization does indeed improve the generalization of the local linear technique, at least for the smooth and non-volatile parent functions investigated here.

Nonetheless, it is worth commenting on how one might choose amongst the variations of this scheme in an algorithmic manner. One obvious way to do this would be to use cross-validation. If the cross-validation is run on the level 1 learning set, then only the parameters dealing with the level 1 generalizer are being varied. The parameters dealing with how to construct the level 1 space (for example) are fixed. Under this scheme we're trying to estimate generalization accuracy in the level 1 space and then use that information to improve the entire structure's generalization of the level 0 learning set. This scheme is equivalent to simply introducing another level (level 2) to the stacking of the generalizers. There is another way to run the cross-validation however; treat the entire stacked generalization process as a generalizer of the level 0 learning set, with a different generalizer corresponding to each different set of parameter values. (Under this scheme we're examining <u>all</u> parameters, including, for example, those dealing with how to map from the level 0 space to the level 1 space.) Now run cross-validation over <u>that</u> set of generalizers. This way we're using the cross-validation to directly estimate generalization accuracy in the level 0 space, which is, after all, what we're ultimately interested in. With this second scheme, the output information going into the level 2 learning set is coming from the level 0 learning set, not the level 1 learning set.

ii) Experiment two

The other numerical experiment was based on the NETtalk "reading aloud" problem. The parent function for this problem has 7 (suitably encoded) letters as input. The output of the parent function is the phoneme that would be voiced by an English speaker upon encountering the middle letter if all 7 letters had occurred in the midst of some text which the speaker was reading aloud. (See (Stanfill and Waltz 1986, Wolpert 1990b, Carterette and Jones 1974, Sejnowski and Rosenberg 1988).) The data set used for the experiment reported here was standard Carterette and Jones (1974), modified (as in (Wolpert 1990b)) to force consistency amongst the several speakers recorded.

In both (Wolpert 1990b) and (Sejnowski and Rosenberg 1988) generalizers never guess directly from 7-letter fields to phonemes. Rather each possible phoneme is decomposed into a vector in a 21-dimensional space (the components of which relate to the physical process of speech). Therefore NETtalk, for example, is a neural net which takes (a suitable encoding of) a 7-letter input field as its input, and guesses a vector in a 21-dimensional space. This vector guess is then converted into a phoneme guess by finding the legal phoneme vector making the smallest angle (in the 21-dimensional space) with the guessed vector. To use metric-based HERBIEs for this problem (as in (Wolpert 1990b)), 21 such HERBIEs have to be used, one for each component of the phoneme vector space. As with the output neurons of NETtalk, the guesses of these 21 metric-based HERBIEs are then passed through a post-processor which combines them to form a 21-dimensional guess, which in turn specifies a phoneme guess. Unless otherwise specified, in the rest of this section, whenever the term "metric-based HERBIE" is used, what is really meant is a set of 21 such HERBIEs combining in the manner discussed here to guess a legal phoneme.

Several separate generalizers were combined in the exact same manner as in the example in section I. Each such level 0 generalizer was a metric-based HERBIE, where 4 nearest neighbors were used. Each of these level 0 generalizers looked exclusively at a different one of the 7 input letter slots, i.e., for each of them instead of using the full Hamming metric $d(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{7} (1 - \delta(\mathbf{p}_i, \mathbf{q}_i))$, the metric $d(\mathbf{p}, \mathbf{q}) = 1 - \delta(\mathbf{p}_k, \mathbf{q}_k))$ for some fixed value of k was used. The level 0 generalizers differed from each other in which letter slot they looked at, i.e., they used different k's. (Effectively, this means that each of the level 0 generalizers had a different one-dimensional input space rather than a seven-dimensional one, since only variations in the k'th slot had any effect on the guessing of the corresponding generalizer).

Three level 0 generalizers were used; the first looked exclusively at the 3rd letter slot of the 7 letter input field, the second looked exclusively at the 4th letter slot, and the third looked exclusively at the 5th letter slot. As in the example in section I, the CVPS was used, the guesses of the level 0 generalizers formed the inputs of the level 1 space, and the outputs of the level 0 and level 1 spaces were identical (i.e., the level 1 output space wasn't an error space). Although it might help the level 1 generalizer if the 21-dimensional output vectors of the level 0 generalizer were fed into the level 1 input space, for simplicity the full level 0 generalizers were used instead and a single integer representing the closest phoneme to the 21-dimensional vector was fed into the level 1 input space. In other words, level 1 inputs were symbolic and not real-valued. The level 1 generalizer was a metric-based HERBIE using a full Hamming metric over the 3-dimensional level 1 input space. (As usual, there were in fact 21 such HERBIEs, making a 21-dimensional guess which in turn specified the phoneme guessed by the entire stacked generalizer.)

The (level 0) learning set was made by looking at successive 7-letter windows of the first 1024 words of Carterette and Jones, i.e., it consisted of $(1024 \times 5) - 6 = 5114$ elements. The testing set was constructed from the successive 7-letter windows of the next 439 words of Carterette and Jones, i.e., it consisted of $(439 \times 5) - 6 = 2189$ elements The three level 0 generalizers got a total of 507, 1520, and 540 correct, respectively, on the testing set. Since each guess was either correct or incorrect, these numbers suffice to determine exactly the expected error in the associated estimates of the average guessing accuracies: generalizer 1 had an average generalizing accuracy of 23% +/- .90%, generalizer 2 had an average generalizing accuracy of 69% +/- .98%, and generalizer 3 had an average accuracy of 25%+/- .92%. As one would expect, generalizer 2, looking at the middle letter of the input field, guesses best what phoneme should correspond to that middle letter.

The stacked generalizer got 1926 correct, for an average accuracy of 88% +/- .69%. Cross-validation (i.e., a level 1 generalizer which worked by globally fitting a surface of the form {level 1 output = one of the level 1 inputs}), would have chosen generalizer 2. Therefore the improvement over cross-validation which resulted from using a better level 1 generalizer was approximately 20 (of generalizer 2's) standard deviations. As in the surface-fitting experiment presented earlier, presumably one could construct a stacked generalizer for the text-to-phoneme problem which performed better than the one presented here. This would be done by varying the parameters of the stacked generalizer, perhaps using a different level 1 generalizer, etc.

The purpose of this text-to-phoneme experiment wasn't to beat the performance (reported in (Wolpert 1990b)) of a metric-based HERBIE having access to all 7 input letters, nor even to beat the performance of back-propagation (i.e., NETtalk) on this data. Rather it was to test stacked generalization, and in particular to test whether stacked generalization can be used to combine separate pieces of incomplete input information. Since some of the letter slots are more important than others for determining the correct phoneme output, this experiment demonstrates stacked generalization's ability to distinguish between (and properly exploit) relevant and (relatively) irrelevant level 0 input information.

III. DISCUSSION OF STACKED GENERALIZATION

There are a number of subtle issues involved with stacked generalization. This section is an introduction to some of them. First two potential shortcomings of stacked generalization are addressed, then a heuristic discussion on the behavior of stacked generalization is presented, and then extensions and variations of the technique are discussed.

i) Multi-valuedness and learning set reproduction

Consider the first example given in section I of how to use stacked generalization, the one involving a set of several $\{G_j\}$. The whole process outlined in that example is itself a generalizer; it takes (level 0) learning sets and (level 0) questions and maps them to guesses. Therefore it's sensible to ask how the whole process agrees with the theoretical properties which are often required of individual level 0 generalizers.

One of the first things one notices is that it's possible for the level 1 learning set to be multi-valued, i.e., the level 1 learning set might contain a pair of points with identical input components but different output components. This is because there might be two (or more) partitions in the partition set which result in the same guesses by all of the $\{G_j\}$ even though they have different θ_{i2}'s. In practice this occurs very rarely, especially if the data takes on a continuum of values. Moreover, unless the level 1 generalizer tries so hard to reproduce its learning set that it can't deal gracefully with such multi-valuedness, this multi-valuedness is not, of itself, a reason for concern. And even if the level 1 generalizer does have a marked lack of grace under such conditions, if the level 1 input space is enlarged to include the level 0 question (or input space projection of θ_{i2}, as the case might be), then the level 1 learning set will now be single-valued and no problems will arise.

Another example of a peculiar property of stacked generalization concerns the issue of reproducing the level 0 learning set. Most conventional generalizers either always reproduce their learning set or strive to do so. However this isn't necessarily the case with the whole process of stacked generalization (viewed as a generalizer of the level 0 learning set), regardless of whether or not the constituent level 0 and level 1 generalizers necessarily reproduce their learning sets. This lack of reproducing the learning set might not be a problem. For example, when one has noisy data, exact reproduction of the learning set is rarely desirable. (Indeed, the behavior of a stacked generalizer when one has large learning sets can perhaps be used as means of determining whether or not one's data is noisy.) And for non-noisy data, it should often be the case that if the learning set is large enough, then the learning set is reproduced, to a good approximation.

Nonetheless, there are many cases where one would like to enforce exact reproduction of the learning set. There are several ways to achieve this. The most obvious is to simply place a filter on the questions being fed to the stacked generalizer: if a level 0 question already exists in the (level 0) learning set, bypass the generalizers and answer that question directly from the learning set. (After all, the purpose of stacked generalization is to improve guessing for questions outside of the learning set, not to provide a complicated means of implementing the look-up-table "if the question is in the input projection of the learning set, guess the corresponding output".)

A more elegant scheme has been devised by Gustafson et al. (1990): require that the level 1 surface guessed by the level 1 generalizer contains the line $\alpha\mathbf{e}$, where α runs over the reals and $\mathbf{e}$ is a diagonal vector, that is a vector in $\mathbf{R}^{k+1}$ all k+1 of whose coordinate projections are identical and non-zero (k being the dimensionality of the level 1 input space). Under this scheme, so long as the level 0 generalizers all reproduce the level 0 learning set, then so will the entire stacked generalizer. The reason is that if the level 0 question is contained in the level 0 learning set, then all the level 0 generalizers will make the same guess (namely the output component of the corresponding element of the level 0 learning set), and then this level 1 generalizer will also make that guess.

There are other ways to ensure reproduction of the learning set which don't restrict the level 1 generalizer. For example, one could, so to speak, *teach* the level 1 generalizer to reproduce the level 0 learning set. To do this, for every set θ_{i1} don't simply create the single element of the level 1 learning set corresponding to $\theta_{i2} = \theta - \theta_{i1}$. Rather for each θ_{i1} create m points in the level 1 space, one for all m possible values of θ_{i2} (i.e., allow θ_{i2} to range over all θ rather than just over $\theta - \theta_{i1}$). Modulo any issues of multi-valuedness of the level 1 learning set, so long as the individual level 0 and level 1 generalizers reproduce their learning sets, then under this scheme so will the entire stacked generalizer.

There are many other criteria one might require of a generalizer in addition to reproduction of the learning set. For example, one might require that the generalizer be invariant under Euclidean symmetry operations in the level 0 space $\mathbf{R}^{n+1}$ (see (Wolpert 1990c)). In practice, although many of the generalizers commonly used reproduce learning sets, few meet these additional generalization criteria (despite the reasonableness of these criteria). The result is that modifying stacked generalization to necessarily obey these criteria is a non-trivial exercise, since in practice the constituent generalizers will often violate them. A full discussion of this and related issues concerning stacked generalization and generalization criteria is beyond the scope of this paper.

ii) Heuristics concerning the behavior of stacked generalization

This sub-section is a cursory heuristic examination of those properties of the individual level 0 and level 1 generalizers which have particularly pronounced effects on the efficacy of stacked generalization.

Many generalizers are explicitly local, meaning the guess they make is overtly dependent in a very strong manner on the nearest neighbors of the question in the learning set. Many other generalizers, while not explicitly local, act locally. For example, back-propagation behaves somewhat locally (see (Lapedes and Farber 1988) and the discussion in (Wolpert 1990e) on using back-propagation to try to generalize the parity input-output function).

Care must be taken whenever one is using such a local generalizer with a CVPS, especially when one is using that generalizer by itself (i.e., when that generalizer is the only level 0 generalizer). The reason is that for several of the i values, the element $\theta_{i2} = \theta - \theta_{i1}$ is one of the elements of θ which lie closest to the level 0 question. Therefore in trying to determine and correct for the biases the level 0 generalizer will have when generalizing with the full learning set θ, training is being done on learning sets with different nearby elements from the nearby elements in the full learning set. However the generalizing of the local generalizer is strongly dependent on the set of nearby elements of the learning set, by hypothesis. Accordingly, the information in the level 1 learning set can be extremely misleading in how it implies the level 0 generalizer will err when answering the level 0 question via the full level 0 learning set.

The natural way to get around this problem is to have the level 1 input space contain information on the nearby elements of level 0 learning set. That way the dependence on the nearby elements of the learning set is being learned. This is exactly the strategy that was followed in the example in section I and the first experiment in section II.

There exist other situations where care must be exercised in using stacked generalization. For example, when the level 1 inputs are given by the outputs of the level 0 generalizers (as in the first example in section I), poor choice of the level 1 generalizer can actually result in generalization performance worse than that of the level 0 generalizers run by themselves. A full characterization of the desirable traits of level 1 generalizers for this situation isn't yet in hand, but some broad observations can be made. In this context, when the level 1 generalizer is explicitly a surface-fitter, best behavior accrues (usually) when that generalizer is relatively global, non-volatile and smooth, and not overly concerned with exact reproduction of the level 1 learning set. For example, in the ongoing work with Lapedes and Farber mentioned at the beginning of section II, the level 1 input space is only 3-dimensional. Nonetheless, the best level 1 generalizers found so far operate by performing what is essentially a uniformly weighted average over several *hundred* of the nearest neighbors of the level 1 question. An even more extreme example is any use of stacked generalization where the level 1 generalizer is a global hyperplane fitter (e.g. cross-validation).

The following simplistic analysis shows why it's reasonable that "relatively global, smooth ..." level 1 generalizers should perform well. Imagine we have a single level 0 generalizer, G, and the level 1 input space is the guess of G. The level 1 outputs are the same as the level 0 outputs. For simplicity, have both the level 0 and level 1 output spaces be discrete-valued with values {1, 2, 3, ..., s}. Let the level 1 generalizer be H. Now assume that G is a fairly good generalizer for the parent function under consideration. More precisely, assume that independent of what guess G makes, that guess is correct exactly 70% of the time. Furthermore, assume that there is only way that G can be fooled for a given guess, i.e., whenever G guesses a particular $t \in \{1, 2, 3, ..., s\}$, then the correct guess is always either t or some other particular number $v_t \in \{1, 2, 3, ..., s\}$ which is determined uniquely by t. Now look at a level 1 input space value $x \in \{1, 2, 3, ..., s\}$. Assume there are $p > 0$ points in the level 1 learning set with this input coordinate. Define $P(\alpha, \beta = p - \alpha)$ as the probability that α of the p elements have output x (meaning G guessed correctly) and β of them have output v_x. $P(\alpha, \beta) = (.7)^{\alpha}(1 - .7)^{\beta} [p! / (\alpha! \beta!)]$, and is independent of the value of G's guess (i.e., this probability is independent of the level 1 input value x). Now because G is correct 70% of the time, in the absence of additional information one should always guess G's output. However if $\beta > \alpha$, then most level 1 generalizers presented with the question x would guess v_x rather than x. (This is especially true if $\alpha = 0$, in which case there's no evidence at all that one should guess anything other than v_x when presented with x.) Assume H has this behavior. Then if p is small for all level 1 input space projections of the level 1 learning set, $P(\alpha, \beta > \alpha)$ is sizable for all those projections of the level 1 learning set. As a result it is likely that a relatively large fraction of those level 1 learning set input space projections have $\beta > \alpha$, i.e., a relatively large fraction of the times H is presented with a question which exists in the level 1 learning set that learning set will lead H to guess v_x rather than G's guess, x. Therefore using stacked generalization with G feeding the level 1 generalizer H will lead to worse guesses than simply using G directly, on average.

This type of problem can occur even if the level 1 input space is multi-dimensional. One simple way around it is to modify H to implicitly estimate G's overall guessing accuracy and then make use of this estimate. To make such an estimate, the level 1 generalizer must examine a large number of the elements of the level 1 learning set and run a cross-validation-type procedure over them (i.e., measure the fit of the hyperplane {output = G's guess} over those elements). Preferably, these examined elements are nearby elements of the level 1 question x, so we don't have to worry about the fact that in general G's guessing accuracy might depend on the value of G's guess (unlike in the toy example above). Putting these requirements together, we get level 1 generalizers which are "relatively global, non-volatile and smooth, and not overly concerned with exact reproduction of the level 1 learning set". Such level 1 generalizers can be viewed as systems which, in effect, boost α (i.e., the number of times G is observed to be right) and β (i.e., the number of times G is observed to be wrong) by examining many nearby elements of the level 1 learning set. With α and β boosted in this way, $P(\alpha, \beta > \alpha)$ becomes small, and our problem is rectified.

Generically, when the level 1 inputs are given by the outputs of the level 0 generalizers, one wants those generalizers to (loosely speaking) "span the space of generalizers" and be "mutually orthogonal" in that space. For example, imagine we have two level 0 generalizers, A and B, whose guesses directly give us the level 1 inputs (see figure 3). Say A is a good generalizer for the parent function, whereas B is not a particularly good generalizer for that function. Then the only possible advantage of using B along with A in a stacked generalization structure is if B adds information not provided by A, i.e., if the correlation between a correct output and the pair {A's guess, B's guess} is greater than the correlation between a correct output and the singlet {A's guess}. If this isn't the case, then B will simply be a red herring, whose guess is redundant with A's (at best). It is for these kinds of reasons that the level 0 generalizers should be "mutually orthogonal".

Similar reasoning justifies the statement that one wants the level 1 generalizers to "span the space". It is usually desirable that the level 0 generalizers are of all "types", and not just simple variations of one another (e.g., we want surface-fitters, Turing-machine builders, statistical extrapolators, etc., etc.). In this way all possible ways of examining the learning set and trying to extrapolate from it are being exploited. This is part of what is meant by saying that the level 0 generalizers should "span the space". Such spanning is important because stacked generalization isn't just a way of determining which level 0 generalizer works best (as in cross-validation), nor even which linear combination of them works best (as in Gustafson et al.'s scheme); rather stacked generalization is a means of non-linearly combining generalizers to make a new generalizer, to try to optimally integrate what each of the original generalizers has to say about the learning set. The more each generalizer has to say (which isn't duplicated in what the other generalizer's have to say), the better the resultant stacked generalization.

Another aspect of what "span the space" means is made clear from the discussion at the very beginning of this sub-section concerning the heuristics of stacked generalization with a single, local generalizer: we would like the output values of the level 0 generalizers to give us all the salient information concerning the nearby elements in the level 0 learning set. These generalizers should collectively tell us all that's important about the level 0 learning set, since otherwise the mapping from the level 0 space to the level 1 space has involved a loss of important information.

Stacked generalization is often nothing more than applying a non-linear transformation to the elements of the learning set before generalizing from them (with the level 1 generalizer). (The non-linear transformation is determined by what level 0 generalizers are used, how they map to the level 1 space, etc.) Saying that the generalizers should be "mutually orthogonal and span the space" essentially means that on the one hand that non-linear transformation should preserve all the important information in the learning set, while at the same time, it should not preserve the redundant and irrelevant information in the mapping from the level 0 space to the level 1 space.

iii) Extensions and variations

There are many interesting implementations of the basic idea of stacked generalization. First, note that the idea of having the level 1 output be an error estimate of a level 0 generalizer G can be applied even when there are other level 0 generalizers in addition to G, all feeding into the level 1 input space. In this case the outputs of the other generalizers are now providing us with information concerning the likely error of G when generalizing from θ. There are a number of advantages to such schemes where the level 1 output isn't interpreted as a guess but rather as an estimate of the error in a guess. For example, with such a scheme the dimensionality of the level 1 input space can be reduced by one without losing any information. (G need no longer feed into the level 1 space to get information concerning G's guess - that information comes in when we subtract the estimated error from G's guess.) Moreover, this scheme allows us to be "conservative"; we can multiply the error estimate by a fraction before subtracting it from G's guess. In this way we can directly control a parameter (the multiplicative fraction) which determines to what extent we use stacked generalization and to what extent we simply use G by itself.

As another interesting implementation, since a stacked generalization structure is itself a generalizer, the whole thing can be stacked, and these stackings can be combined into a network structure.[7] All the usual net games (e.g. back-propagation) can then be applied to this network structure. Another interesting variation is to have the level 0 generalizers all be similar, relatively dumb systems An example of such a system is the following generalizer: "guess the output value of the point in the learning set whose input component lies closest to the vector sum of some fixed input space vector with the question". Different level 0 generalizers have a different "fixed input space vector". (If that ""fixed input space vector" = 0, then we recover the traditional nearest neighbor generalizer.) Non-linear time-series analysis, with its "delay embedding" (Farmer and Sidorowich 1988, Casdagli 1989), is an example of such a use of stacked generalization with a set of similar, dumb, level 0 generalizers.[8] Other examples of this kind of implementation of stacked generalization are fan generalizers (Wolpert 1990e), the extension of non-linear time-series analysis to multiple input dimensions.[9]

Other variations are interesting as tools for theoretical investigations of stacked generalization. For example, let N, the number of generalizers, equal n, the dimension of the level 0 input space, and also use a partition set in which r = m. Use multiple stacking and have the level k space's inputs be the outputs of the N level (k - 1) generalizers, exactly as in the example in section I (where k = 1). For this implementation of stacked generalization, when producing the learning set one level above it, any generalizer, <u>no matter what level it's working at</u>, reduces to a single unique function g_{m-1}, taking as argument an n-dimensional question and an (m - 1)-element learning sets whose input space is n-dimensional. As a result, we can explicitly analyze the behavior of the whole system as more and more stacking levels are added. For example, we can consider the case where each level has a single learning set, and all such learning sets feed serially into the set one level above, all according to the exact same rules. (Such a structure is a multi-layer net, where each node is a learning set, there exists one node per layer, and information is fed from one node to the next via the N generalizers.) For such a scenario the successive levels act upon the learning set like successive iterations of an iterated map. Therefore the usual non-linear analysis questions apply: when does one get periodic behavior? when does one get chaotic behavior? what are the dimensions of the attractors? etc. Once answered, such questions would presumably help determine how many levels to stack such a system.

Yet another interesting theoretical scenario arises when not only can all the mappings from one learning set to the next be reduced to a single function g_{m-1}, but so can the guessing for questions outside of the learning set(s). This full reduction usually doesn't obtain due to the fact that the cardinality of θ_{i1} is less than the cardinality of the full θ, and therefore a question $\not\in \theta$ goes through a different g than θ_{i2} (g_m vs. g_{m-1}). (The same conundrum arises when trying to provide theoretical justifications for techniques like cross-validation.) One obvious way around this difficulty is to have g_m fixed by g_{m-1}. For example, one could define $g_m(\theta; q) \equiv < g_{m-1}(\theta_{i1}; q) >_{\{i\}}$, where the θ_{i1} are chosen from the CVPS of θ. (The averaging can either be done with a uniform weighting over all m numbers $g_{m-1}(\theta_{i1}; q)$, or those numbers might be weighted according to the error value $|g_{m-1}(\theta_{i1}$; input component of $\theta_{i2})$ - (output component of $\theta_{i2})|$.) In this way an analysis of the generalizing behavior of stacked generalization and its relation to the constituent generalizers could be cast in terms of the behavior of a single function.

Finally, it's interesting to note that some authors have investigated what amounts to stacked generalization in the context of improving learning (i.e., improving reproduction of the learning set) rather than improving generalization. In such a context, θ_{i1} can be allowed to run over the entire learning set. An example of such a scheme is investigated in (Deppisch et al. 1990). The level 1 generalizer used in(Deppisch et al. 1990) is back-propagation on standard feed-forward neural nets, and the level 1 output space is the error of the level 0 generalizer. The level 1 input space is identical to the level 0 input space. The level 0 generalizer is also back-propagation on standard feed-forward neural nets, only restricted to have a non-zero resolution in its output. Evidence is presented in (Deppisch et al. 1990) indicating that this scheme achieves much lower learning error than a single back-propagation generalizer, and does so in much less time.

It should be noted however that although a partition set of the type implicitly used in (Deppisch et al. 1990) might help learning, it entails some major disadvantages as far as generalization is concerned. For example, if this partition set is used when there is no noise, and if one of the level

0 generalizers guesses perfectly for questions on which it has been trained, then, *as far as the level 1 generalizer can tell*, that level 0 surface-fitter always guesses perfectly for all questions. Accordingly, any reasonable level 1 generalizer will simply say that one should use that level 0 generalizer directly, and ignore any other level 0 information. In general, when using this partition set one is not "generalizing how to generalize" but rather "generalizing how to learn", in that the level 1 space contains information on how well the level 0 generalizers *learn*, but not on how well they *generalize*.

CONCLUSION

Stacked generalization is a generic term referring to any scheme for feeding information from one set of generalizers to another before forming the final guess. The distinguishing feature of stacked generalization is that the information fed up the net of generalizers comes from multiple partitionings of the original learning set, all of which split up that learning set into two subsets. Each such pair of subsets is then used to glean information about the biases of the generalizing behavior of the original generalizer(s) with respect to the learning set. (Note that this is not the same as the biases of the *learning* behavior of the original generalizer(s).) It is this bias information which is fed up the net; stacked generalization is a means of estimating and correcting for the biases of the constituent generalizer(s) with respect to the provided learning set.

Stacked generalization can be used with a single generalizer, in which case it is explicitly a scheme for estimating and correcting the errors of that generalizer. The surface-fitting experiments reported here indicate that it can be quite effective at correcting those errors. When used with multiple generalizers all of which feed into a single back-end generalizer, certain special cases of stacked generalization are exactly equivalent to cross-validation, certain are exactly equivalent to forming a linear combination of the guesses of the constituent generalizers, etc. All such special cases correspond to the assumption of a particular (invariably rather dumb) back-end generalizer.

As with any other generalizing problem, use of more sophisticated generalizers should be expected to give improved results. This is indeed the case, according to the NETtalk-type experiments reported here and according to other experiments reported elsewhere. The conclusion is that for many generalization problems stacked generalization can be expected to reduce the generalization error rate.

FOOTNOTES

[1] Strictly speaking, the amount of information in the learning set ~ the number of bits defining the set of parent functions consistent with that learning set (see (Anshelevich et al. 1989)). The extra information implicit in stacked generalization comes from the assumption that in-sample/out-of-sample techniques are accurate indicators of generalization behavior for the entire learning set. This assumption is implicit in most non-parametric statistics techniques (e.g. the non-parametric statistics techniques discussed below).

[2] There are no guarantees, of course. Some non-cross-validation schemes for choosing amongst a set of generalizers (e.g. parsimony, or even random choice) will in certain circumstances result in choosing a generalizer which has a lower generalization error rate than the generalizer chosen by cross-validation. Similarly, in certain circumstances some scheme other than stacked generalization (e.g., just using one of $\{G_j\}$ straight, by itself) will outperform stacked generalization. This non-universality is inevitable, and <u>holds for any generalizing scheme whatsoever</u>, due to the fact that guessing a parent function based on only a finite number of samples of it is an ill-posed problem in the Hadamard sense (see (Morozov 1984)).

[3] In addition to stacked generalization, there are other ways of embedding the central idea of cross-validation in a more sophisticated framework. One such is to not use the cross-validation error simply as a means for choosing amongst a set of generalizers. Rather one constructs a generalizer from scratch, requiring it to have zero cross-validation error. (To make the construction unique, one must impose other constraints as well - see (Wolpert 1990d).) Instead of coming up with a set of generalizers and then observing their behavior, one takes the more enlightened approach of specifying the desired behavior first, and then solving the inverse problem of calculating the generalizer with that desired behavior. This approach is called "self-guessing". It is similar in spirit to regularization theory, except that here (loosely speaking) the regularization is being done over the space of generalizers as opposed to the space of input-output functions.

[4] Although Gustafson et al. don't view it in those terms, their scheme is essentially the same as cross-validation, except that instead of finding a single best generalizer, they're finding the best (restricted) linear combination of generalizers, using a CVPS to determine that combination. In the language of stacked generalization, their scheme is using a CVPS along with a level 1 input space consisting of the outputs of the level 0 generalizers. The level 1 output space is the correct outputs from the level 0 space, and the level 1 generalizer is a restricted global hyperplane fitter. (The level 0 generalizers in their scheme are variations of local hyperplane fitters.) The difference between this scheme and straight cross-validation is that the restrictions Gustafson et al. imposes on the level 1 generalizer are more lax. They too generalize in the level 1 space by fitting with a global hyperplane, but they allows arbitrary hyperplanes of the form $\Sigma\, a_i x_i$, where the x_i are the level 1 input space coordinates and the a_i are arbitrary real-valued constants restricted so that $\Sigma\, a_i = 1$. (In contrast, cross-validation adds the extra restriction that all but one of the a_i must equal 0.)

[5] As implemented by He Xiangdong, GMDH can be viewed as using the following partition

set rather than the so-called "GMDH partition set": θ_{i2} ranges over all single pairs from θ, just as in a CVPS, but $\theta_{i1} = \theta$ for all i. There are then p level 0 generalizers, all of which are identical except that they use non-overlapping subsets of θ to train themselves. (p is usually restricted to be a divisor of m.) For example, the first level 0 generalizer might be a surface-fitter which only fits an i/o surface to the group of the first m/p elements of θ, the second level 0 generalizer is the same surface fitter but fits an i/o surface to the second group of m/p elements of θ, and so on. The GMDH scheme of He Xiangdong consists of feeding those p level 0 generalizers into a level 1 space and generalizing there.

[6] More sophisticated versions of metric-based HERBIEs replace a pre-fixed metric with something less restrictive. For example, in the use of metric-based HERBIEs reported in (Wolpert 1990b), the input space was 7 dimensional, and each of the 7 coordinates of any input value were scaled by a distinct weighting factor, ρ_i, $1 \leq i \leq 7$, before the conventional metric was applied. The weighting vector ρ_i was determined by the learning set itself via cross-validation. A more general scheme would be to use a weighting matrix rather than a weighting vector. In this scheme, one multiplies all input space vectors by the weighting matrix before applying the conventional metric. (Use of a weighting vector is a special case of this scheme where the weighting matrix is diagonal.) Again, in practice something like cross-validation could be used to find the matrix. (Since the space of possible matrices is so large however, rather than the trial and error approach used in (Wolpert 1990b) one would probably want to employ something like gradient descent in the space of cross-validation error to find the "optimal" weighting matrix.) Pre-multiplying by a weighting matrix is equivalent to linearly transforming the input space before doing the generalizing. Such a transformation allows cross-talk amongst the various input space coordinates to occur in the determination of distances between input space vectors. This jump from use of a weighting vector to use of a weighting matrix in a metric-based HERBIE is loosely equivalent to the jump from using a perceptron (with its vector of synaptic weights) to using a feedforward neural net with a single hidden layer (where one has matrices of synaptic weights). After all, the mapping from the input layer to

the hidden layer in a neural net is nothing more than a linear transformation of the original input vectors.

[7] This is essentially what is done in (Wolpert 1990d), where a genetic evolution process is used to create a feedback net of generalizers. (In (Wolpert 1990d), the output of this feedback net of generalizers is fed through yet another generalizer to get the final guess. This final generalizer has its learning set constructed so that the original level 0 learning set is reproduced. The learning sets for all the other generalizers are instead determined by the evolutionary development of the net. The fitness function for this evolution is the cross-validation error of the entire system.) One interesting aspect of such nets of generalizers is that one can have an "environment generalizer". One (or more) of the nodes in the net can be reserved for a generalizer whose input-output function serves the same purpose as input lines in more conventional architectures. For example, a one-dimensional input environment, say of brightness vs. angle, is a function. Discretize the independent variable of this function and feed the resultant numbers into the input nodes, one number per node, and you get the conventional way of feeding data into a net. If instead one finds a learning set which, when generalized (by a surface-fitter say) gives you the environment function, then you can insert that generalizer together with that learning set (i.e., that environment function) as an "environment generalizer" node in the net. With this scheme different environments don’t correspond to different values on input lines; they correspond to different environment generalizers at the appropriate nodes of the net. This scheme has the advantages that it allows the net to actively query its environment, and also allows that environment to have arbitrary size. (Neither of these properties hold for the conventional process of discretizing that environment and feeding it into input nodes.) See (Wolpert 1990d) for details.

[8] In conventional univariate non-linear time-series analysis, one is provided a sequence of values of a single-dimensional variable for a set of times: $y(j\tau)$, $1 \le j \le m$, τ some real-valued constant. To try to generalize from the sequence one assumes that the value of y at a time t, y(t), is

determined by its value at a set of p delays, $y(t - \tau)$, $y(t - 2\tau)$, ..., $y(t - p\tau)$. To exploit this assumption one "embeds" the original sequence as a learning set in a space with p dimensions of input and one dimension of output. Each element of this delay-space learning set has its input components set to the values $y(t - \tau)$, $y(t - 2\tau)$, ..., $y(t - p\tau)$ for some sequence of p values chosen from the provided time-series, and the output value of that element is now the value of the point $y(t)$, again read off of that time-series. One has as many points in the delay-space learning set as there are sequences of $p + 1$ consecutive points in the time series. To make a prediction for $y(T)$, given the p values $y(T - \tau)$, $y(T - 2\tau)$, ..., $y(T - p\tau)$, one simply generalizes in the delay space, i.e., one guesses what output should correspond to the delay space question $\{y(T - \tau), y(T - 2\tau), ..., y(T - p\tau)\}$, basing this guess on the delay space learning set. Viewed in terms of stacked generalization, this whole embedding procedure is nothing more than a set of level 0 generalizers feeding into a level 1 generalizer. The level 0 learning set has a one-dimensional input space - it's the time series. The p constituent level 0 generalizers are all predicated on the assumption that that time series is periodic. They differ from one another only in what period they assume for the series; one level 0 generalizer assumes period τ, one assumes period 2τ, etc., all the way up to an assumption of period $p\tau$. (In order, these generalizers work by predicting $y(t) = y(t - \tau)$, by predicting $y(t) = y(t - 2\tau)$, etc.) When presented with the question $y(t)$, the k'th level 0 generalizer in conventional non-linear time-series analysis makes a guess which is completely independent of all of the elements of the level 0 learning set except for $y(t - k\tau)$; quite a dumb generalizer. From this perspective of stacked generalization, one can immediately see an obvious way to try to improve the performance of non-linear time-series analysis: replace the p level 0 generalizers which rigidly assume exact periodicity with periods τ, 2τ, ..., $p\tau$ with generalizers which aren't quite so pig-headed. For example, one could instead use generalizers which only assume that $y(t)$ is *set by* $y(t - \tau)$, that $y(t)$ is set by $y(t - 2\tau)$, etc. All these level 0 generalizers could then use a conventional generalizer (e.g. a metric-based HERBIE) along with the entire (!) time-series to estimate *how* $y(t)$ is set by $y(t - \tau)$, how $y(t)$ is set by $y(t - 2\tau)$, etc. Under this scheme, instead of simply having the k'th level 0 generalizer predict *exactly* $y(t - k\tau)$ when provided with the question $y(t)$, that generalizer guesses what answer should correspond to $y(t)$ *based* *on* $y(t - k\tau)$.

[9] It’s interesting to examine fan generalizers from the point of view of the discussion earlier on "spanning the space of generalizers". Although producing the inputs of the level 1 learning set exclusively from the outputs of the level 0 learning set, fan generalizers nonetheless preserve all the "salient" information about the input space geometry of the level 0 learning set. They do this via the fan itself, which consists entirely of level 0 input space information and is crucial to the construction of the input components of the elements of the level 1 learning set.

REFERENCES

Anshelevich, V.V., et al. (1989). On the ability of neural networks to perform generalization by induction. *Biological Cybernetics*, **61**, 125-128.

Carterette, E.C. and Jones, M.H. (1974). **Informal Speech**. University of California Press, Los Angeles.

Casdagli, M. (1989). Non-linear prediction of chaotic time-series. *Physica D*, **35**, 335-356.

Dietterich, T. G. (1990). Machine learning. *Annual review of computer science*, **4**, 255-306.

Deppisch, J., et al. (1990). Hierarchical training of neural networks and prediction of chaotic time series. *Institut fur Theoretische Physik und SFB Nicktlineare Dynamik, Universitat Frankfurt, Germany.* No report number.

Efron, B. (1979). Computers and the theory of statistics: thinking the unthinkable, *SIAM REVIEW*, **21**, 460-480.

Farmer, J.D., and Sidorowich, J.J. (1988). Exploiting chaos to predict the future and reduce noise, Los Alamos report LA-UR-88-901

Gustafson, S., Little, G., and Simon, D. (1990). Neural network for interpolation and extrapolation. *Report number 1294-40, the University of Dayton, Research Institute, Dayton, Ohio.*

Holland, J. (1975). **Adaptation in natural and artificial systems**. University of Michigan Press.

Lapedes, A., and Farber, R. (1988). How neural nets work, Proceedings of the 187 IEEE Denver conference on neural networks, published in *Neural Information Processing Systems*, D.Z. Anderson (Ed.), 1988, published by the American Institute of Physics.

Li, Ker-Chau (1985). From Stein's unbiased risk estimates to the method of generalized cross-validation, *The Annals of Statistics*, **13**, 1352-1377.

Morozov, V.A. (1984). Methods for solving incorrectly posed problems. Springer-Verlag.

Omohundro, S. (1987). Efficient algorithms with neural network behavior. Report UIUCSCS-R-87-1331 of the University of Illinois at Urbana-Champaign Computer Science Department.

Poggio, T., and staff, MIT AI Lab (1988). MIT progress in understanding images. In L. Bauman (Ed.), **Proceedings of the image understanding workshop**. McLean, VA.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, **1**, 81-106.

Rissanen, J. (1986). Stochastic complexity and modeling. *The Annals of Statistics*, **14**, 1080-1100.

Rumelhart, D. E., and McClelland, J. L. (1986). **Explorations in the microstructure of cognition, volumes I and II**. MIT Press, Cambridge, MA.

Schulz, G. E., et al. (1974). Comparison of predicted and experimentally determined secondary structure of adenyl kinase, *Nature*, **250**, 140-142.

Sejnowski, T.J., and Rosenberg, C. R. (1988). NETtalk: a parallel network that learns to read aloud, *Report No. JHU/EECS-86/01, Johns Hopkins University, Electrical Engineering and Computer*

Science Dept.

Stanfill, C., and Waltz, D. (1986). Toward memory-based reasoning. *Communications of the ACM*, **29**, 1213-1228.

Stone, M. (1977). Asymptotics for and against cross-validation, *Biometrika*, **64**, 29-35.

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

Wolpert, D. (1989). A benchmark for how well neural nets generalize. *Biological Cybernetics*, **61**, 303-313.

Wolpert, D. (1990a). The relationship between Occam's razor and convergent guess. *Complex Systems*, **4**, 319-368.

Wolpert, D. (1990b). Constructing a generalizer superior to NETtalk via a mathematical theory of generalization. *Neural Networks*, **3**, 445-452.

Wolpert, D. (1990c). A mathematical theory of generalization: part I. *Complex Systems*, **4**, 151-200.

Wolpert, D. (1990d). A mathematical theory of generalization: part II. *Complex Systems*, **4**, 200-249. "Cross validation" is a special case of the property of "self-guessing" described in this paper.

Wolpert, D. (1990e). Improving the performance of generalizers via time-series-like pre-processing of the learning set, *Report No. LA-UR-90-401, Los Alamos National Laboratory, NM*. Submitted to IEEE PAMI.

He Xiangdong and Zhu Zhaoxuan (1990). Nonlinear time series modeling by self-organizing methods. *Report from the Department of mechanics, Peking University, Beijing, PRC*. No report number.

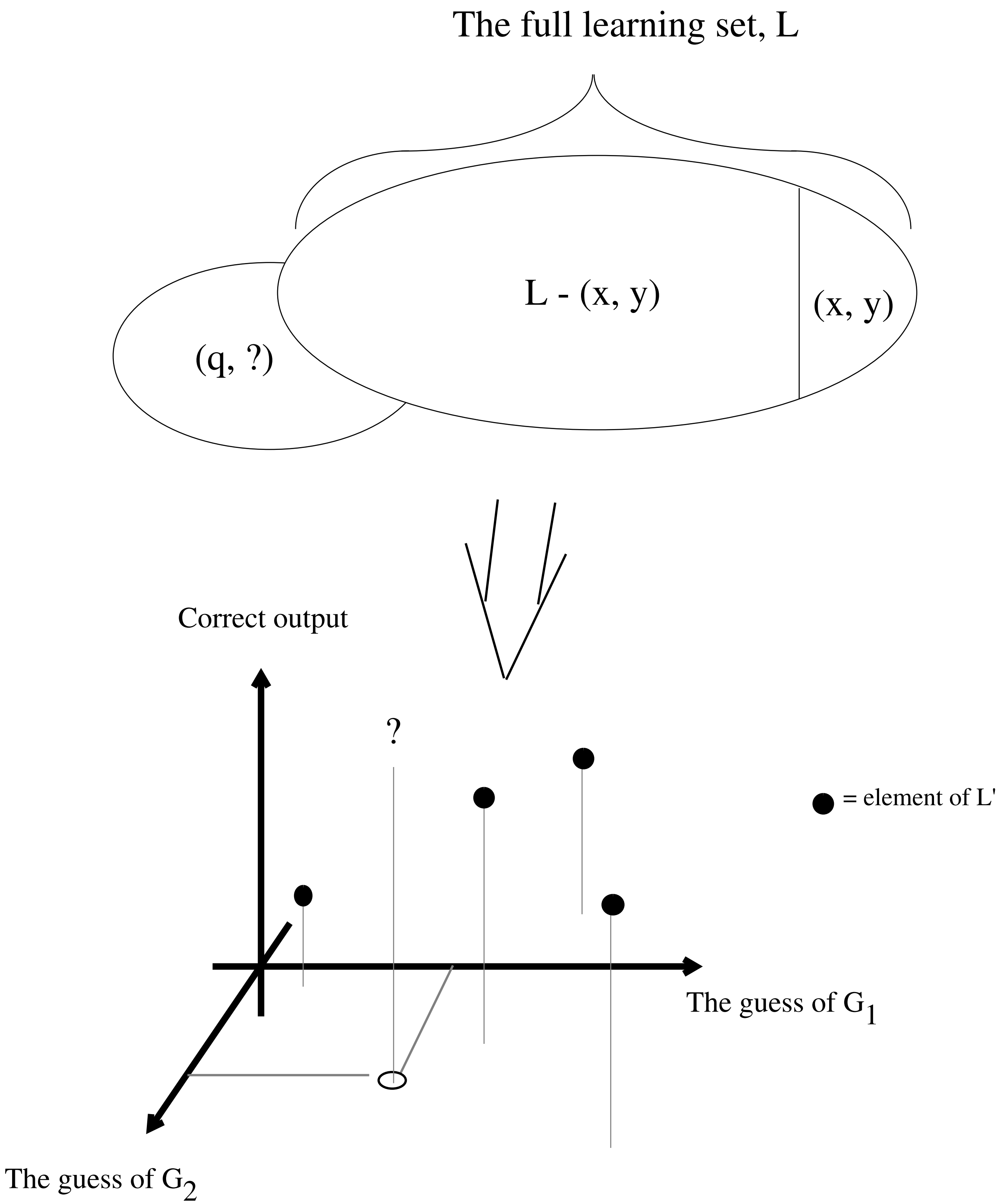

The full learning set, L
L - (x, y)
(x, y)
(q, ?)
Correct output
?
= element of L'
The guess of G1
The guess of G2

Figure 1.

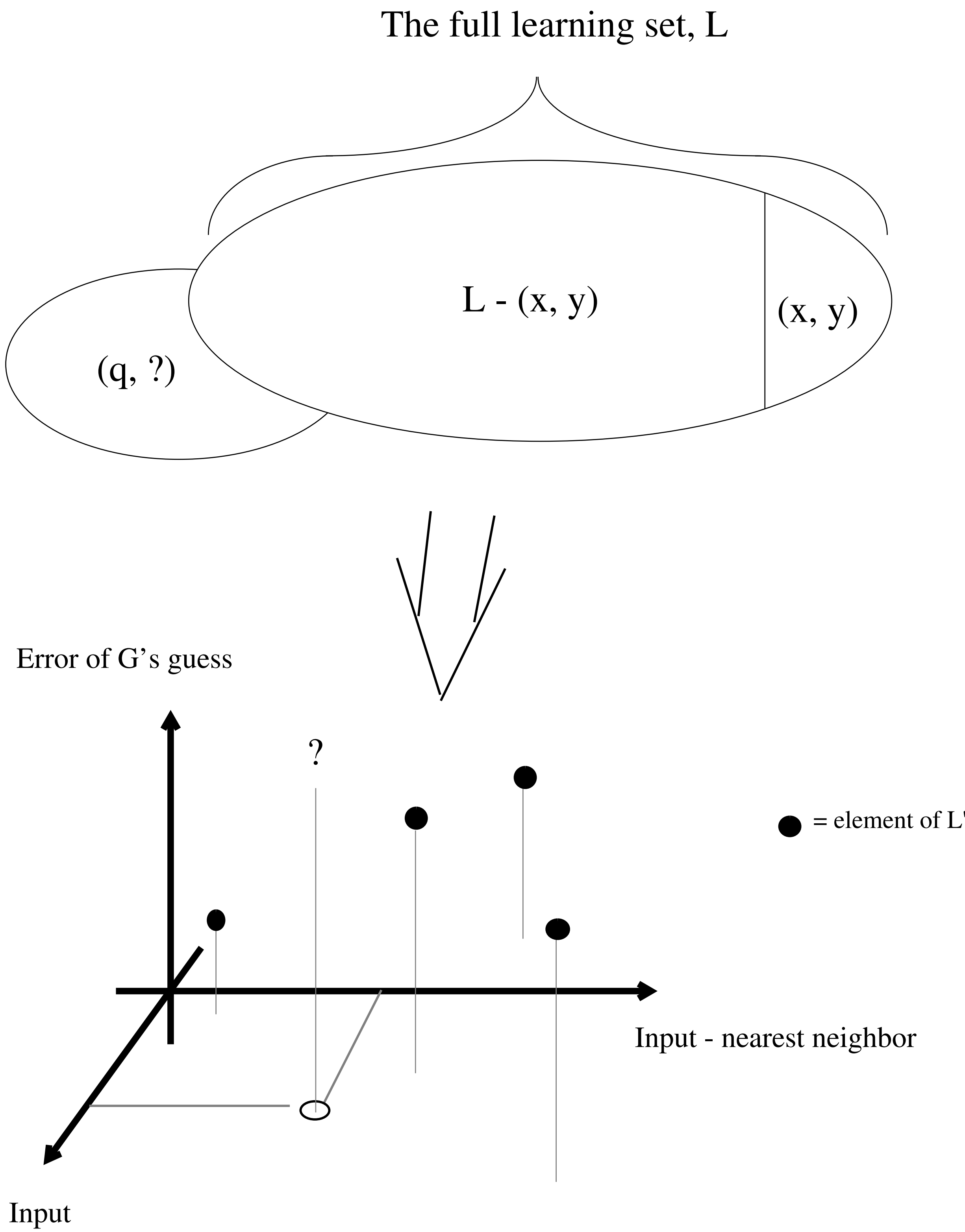

The full learning set, L
L - (x, y)
(x, y)
(q, ?)
Error of G's guess
?
= element of L'
Input - nearest neighbor
Input

Figure 2.

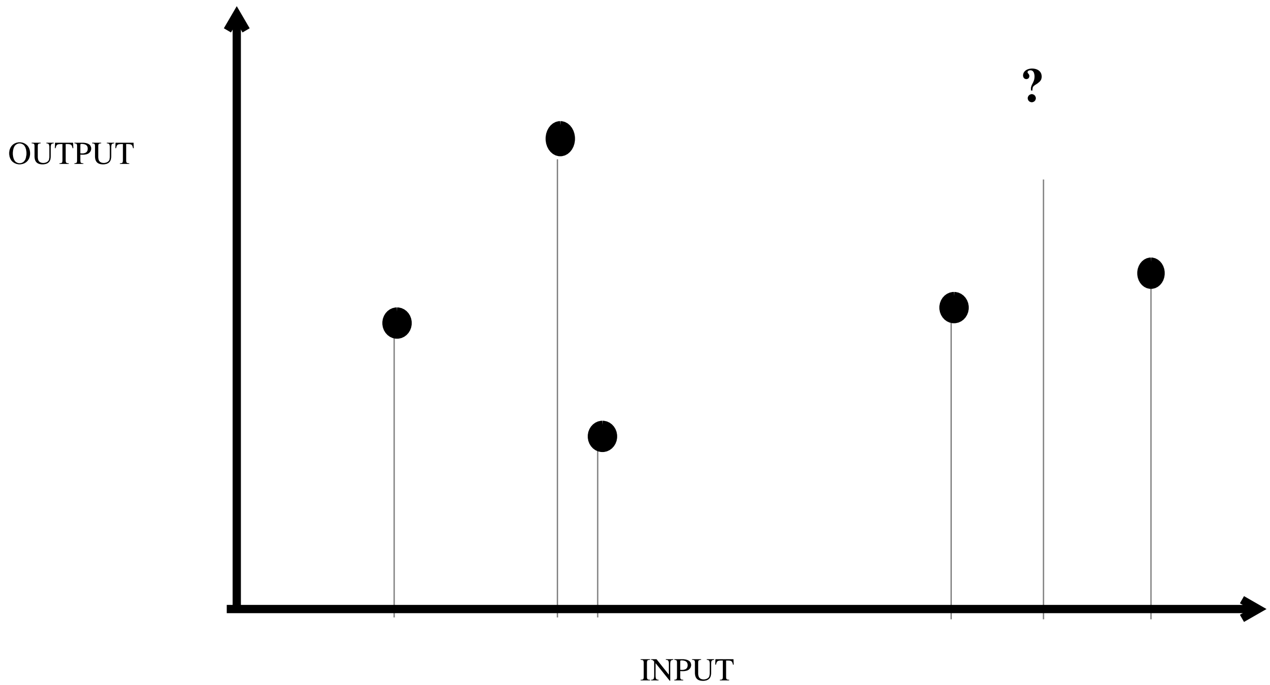


Figure 3a.

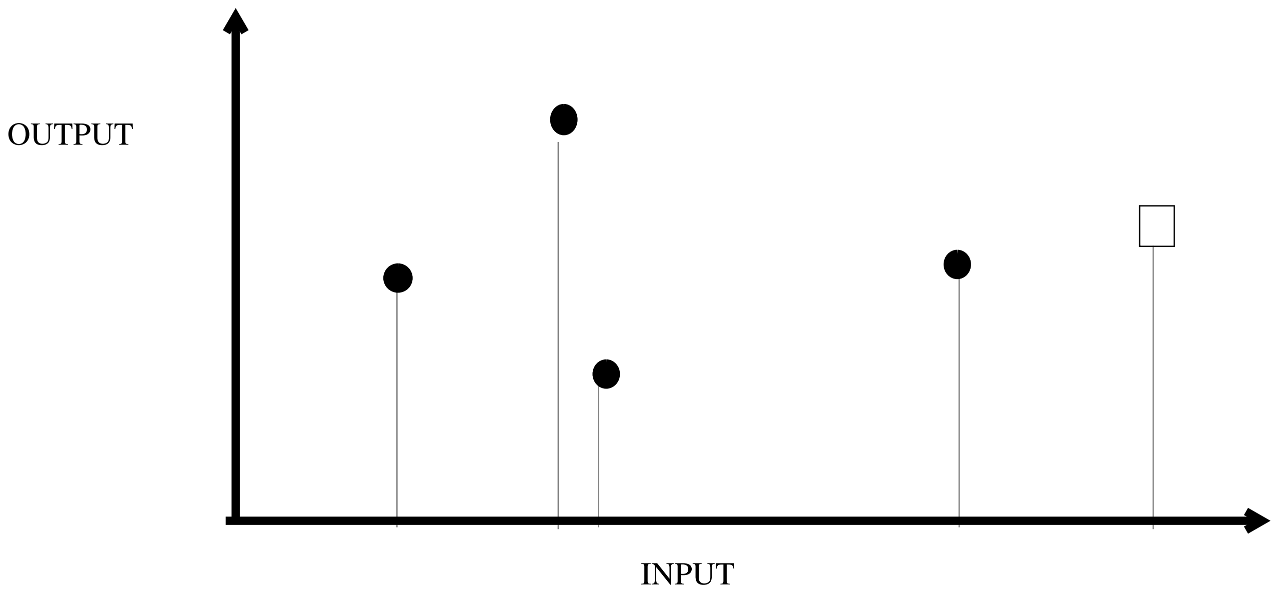


Figure 3b.

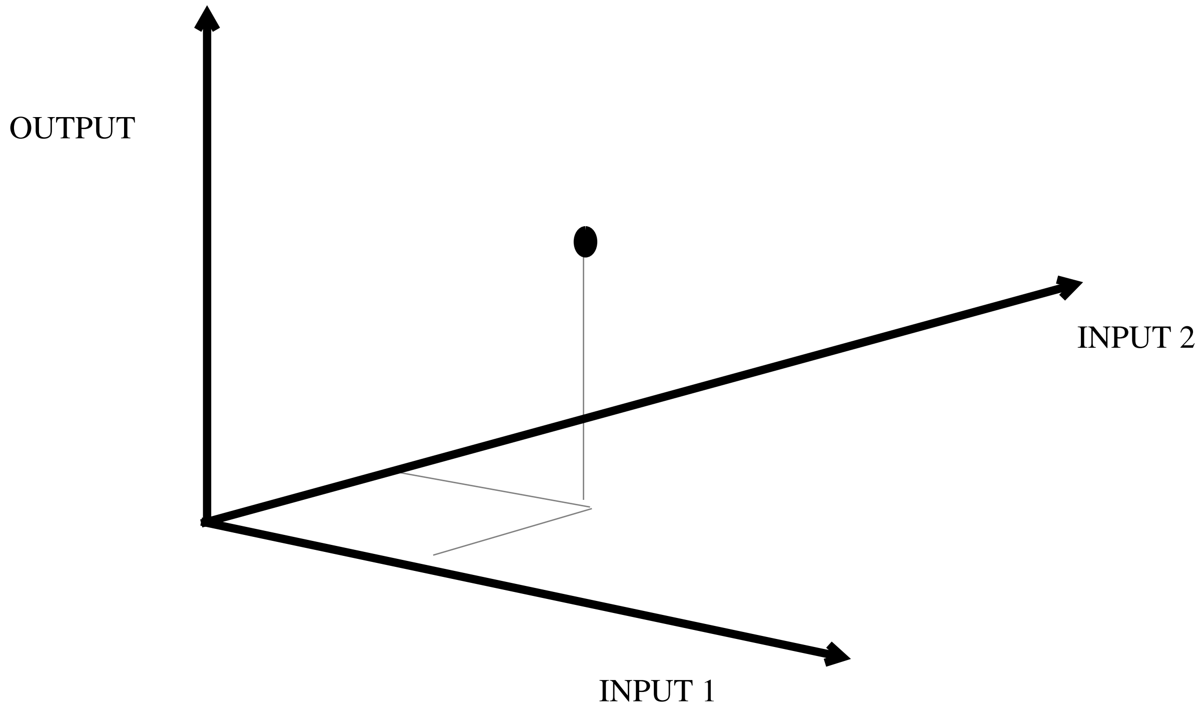


Figure 3c.

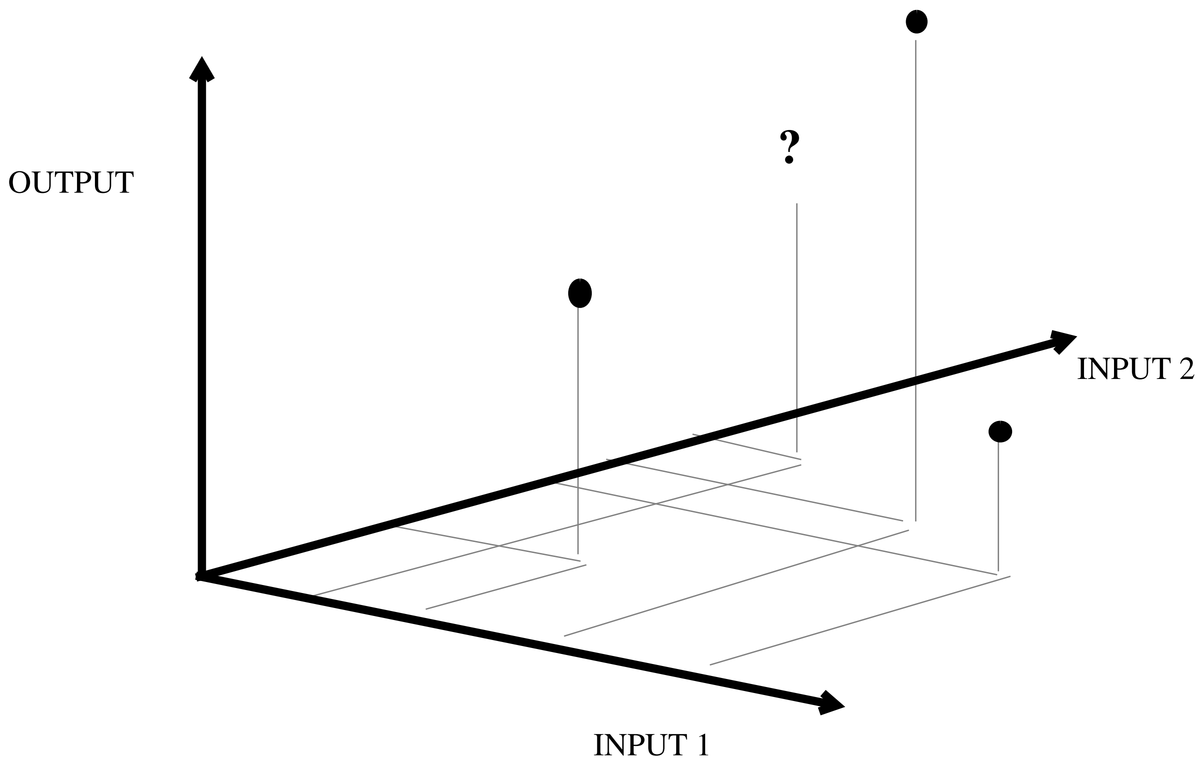


Figure 3d.

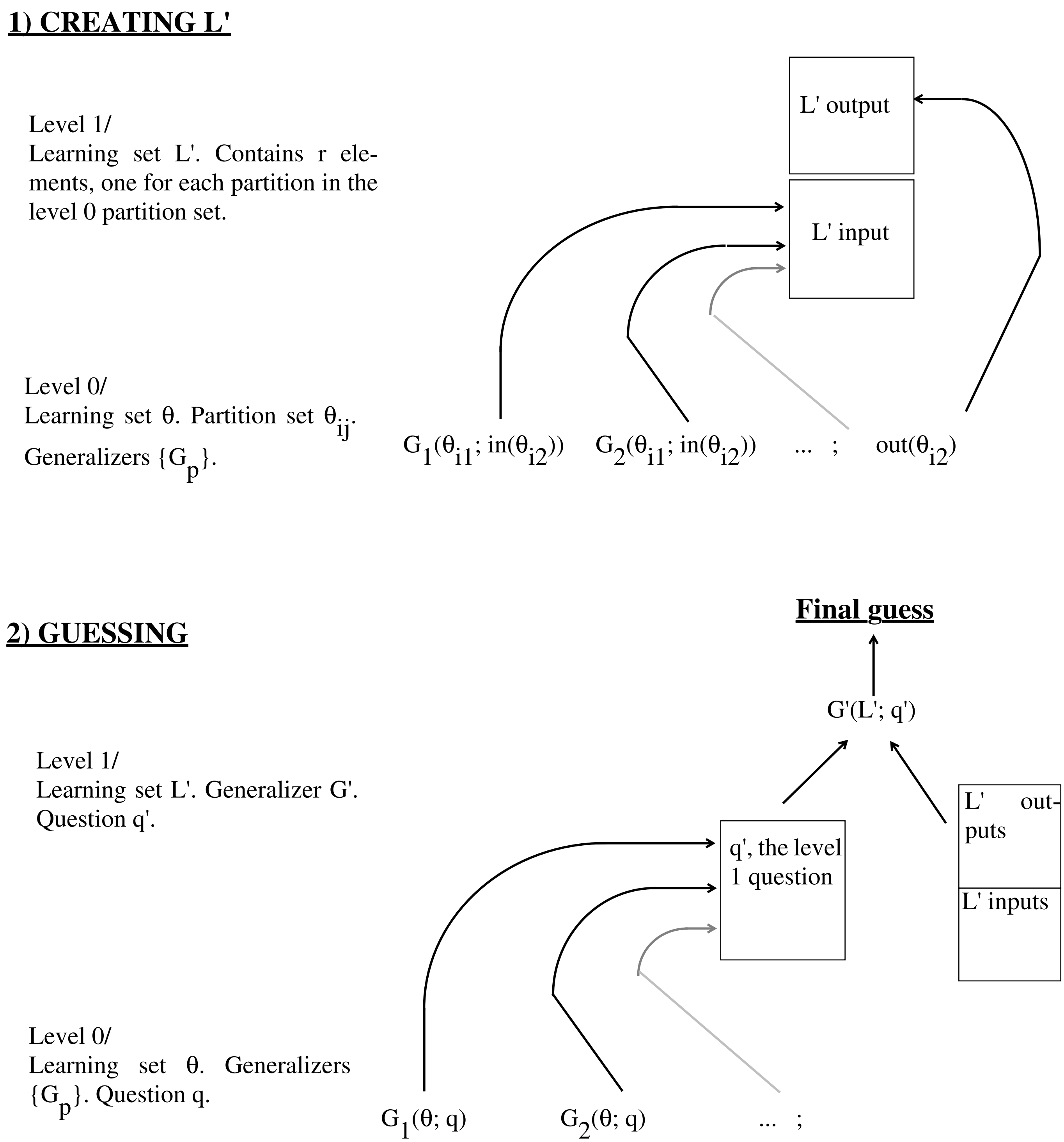

1) CREATING L'
Level 1/
Learning set L'. Contains r elements, one for each partition in the level 0 partition set.
L' output
L' input
Level 0/
Learning set θ. Partition set θ_{ij}.
Generalizers $\{G_p\}$.
$G_1(\theta_{i1}; in(\theta_{i2}))$
$G_2(\theta_{i1}; in(\theta_{i2}))$
... ;
$out(\theta_{i2})$
2) GUESSING
Final guess
G'(L'; q')
Level 1/
Learning set L'. Generalizer G'. Question q'.
q', the level 1 question
L' outputs
L' inputs
Level 0/
Learning set θ. Generalizers $\{G_p\}$. Question q.
$G_1(\theta; q)$
$G_2(\theta; q)$
... ;

Figure 4.

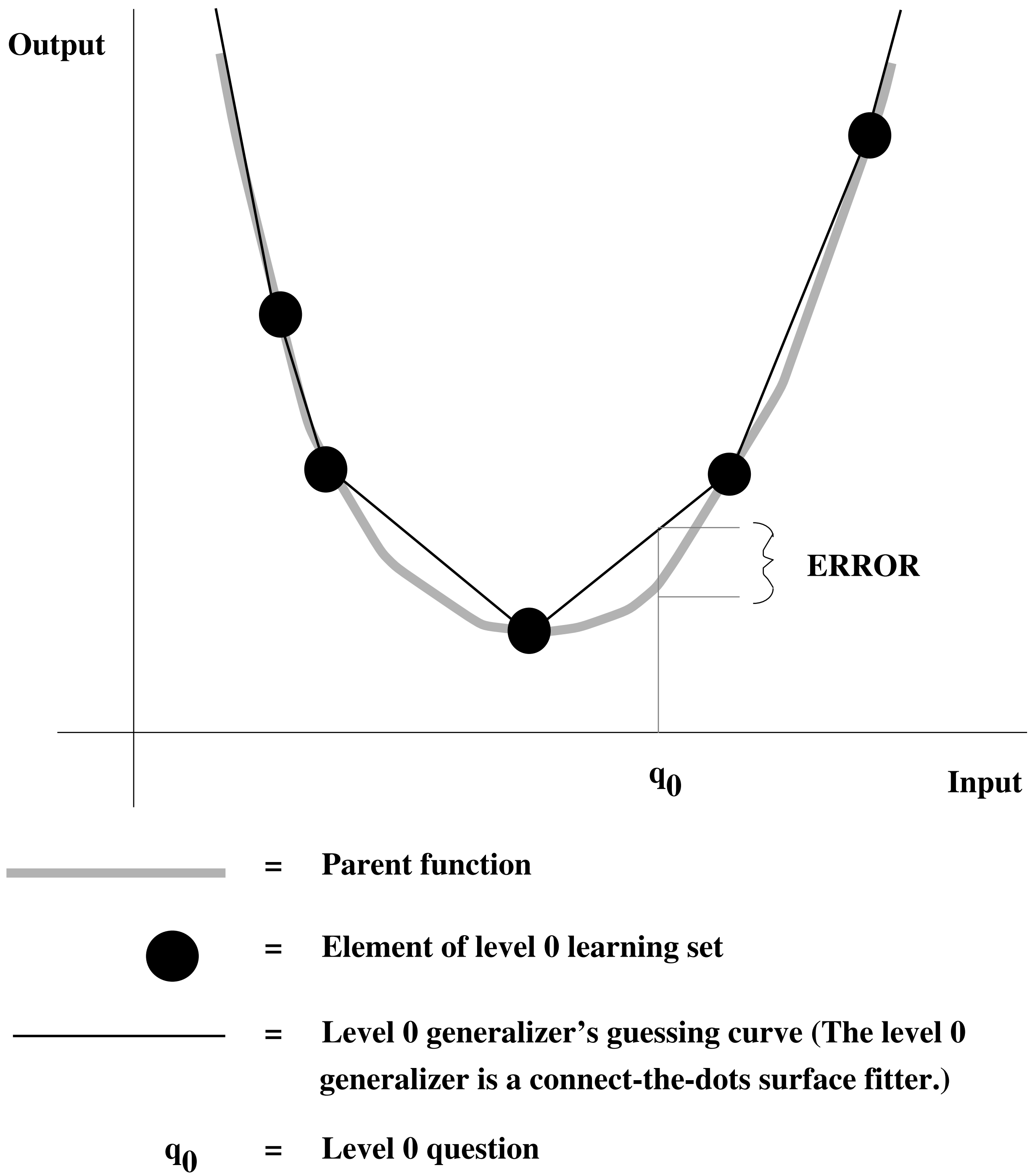
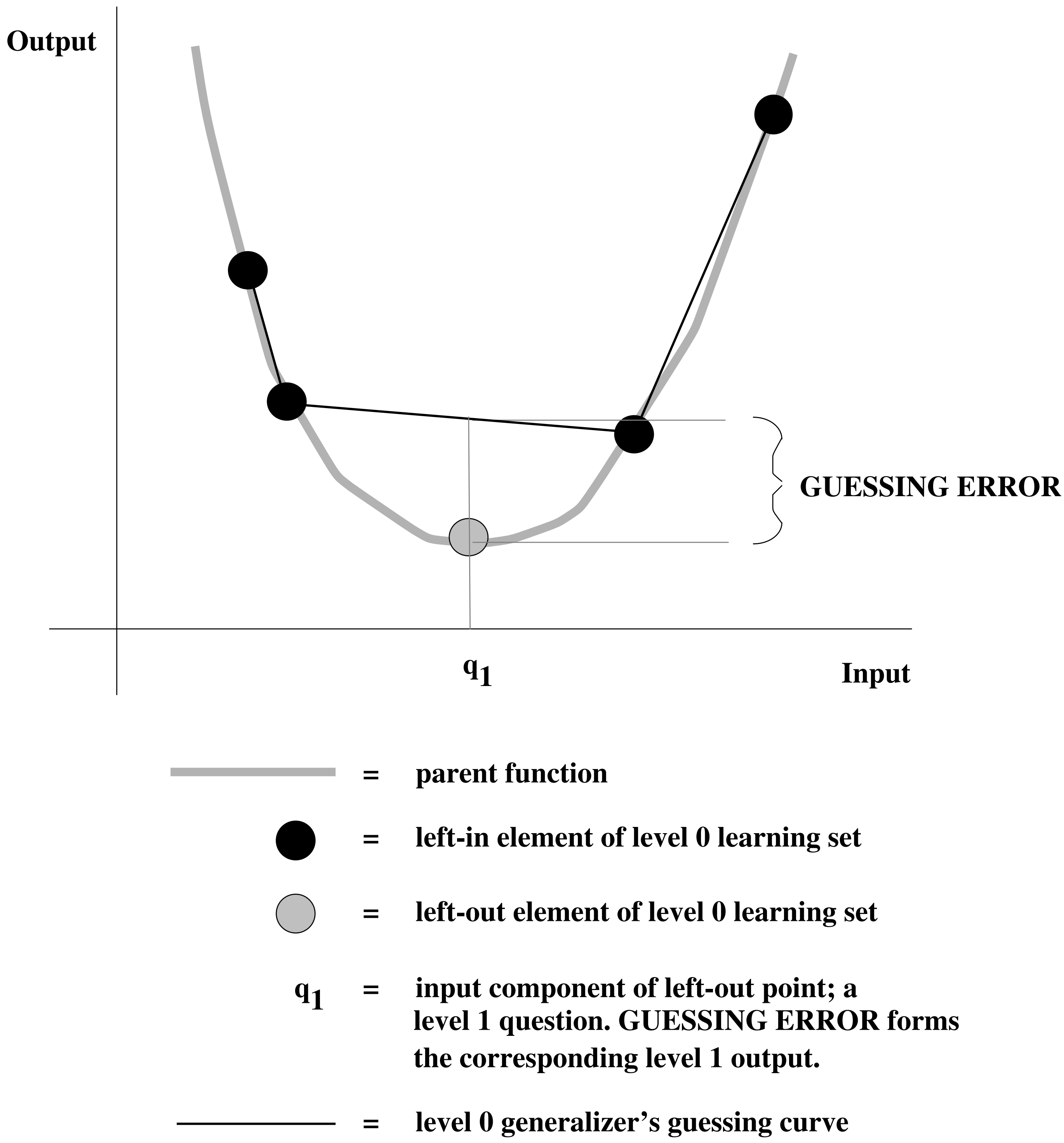

Output
ERROR
q_0
Input
= Parent function
= Element of level 0 learning set
= Level 0 generalizer’s guessing curve (The level 0 generalizer is a connect-the-dots surface fitter.)
q_0 = Level 0 question

Figure 5a.



Output
GUESSING ERROR
q_1
Input
= parent function
= left-in element of level 0 learning set
= left-out element of level 0 learning set
q_1 = input component of left-out point; a level 1 question. GUESSING ERROR forms the corresponding level 1 output.
= level 0 generalizer’s guessing curve

Figure 5b.

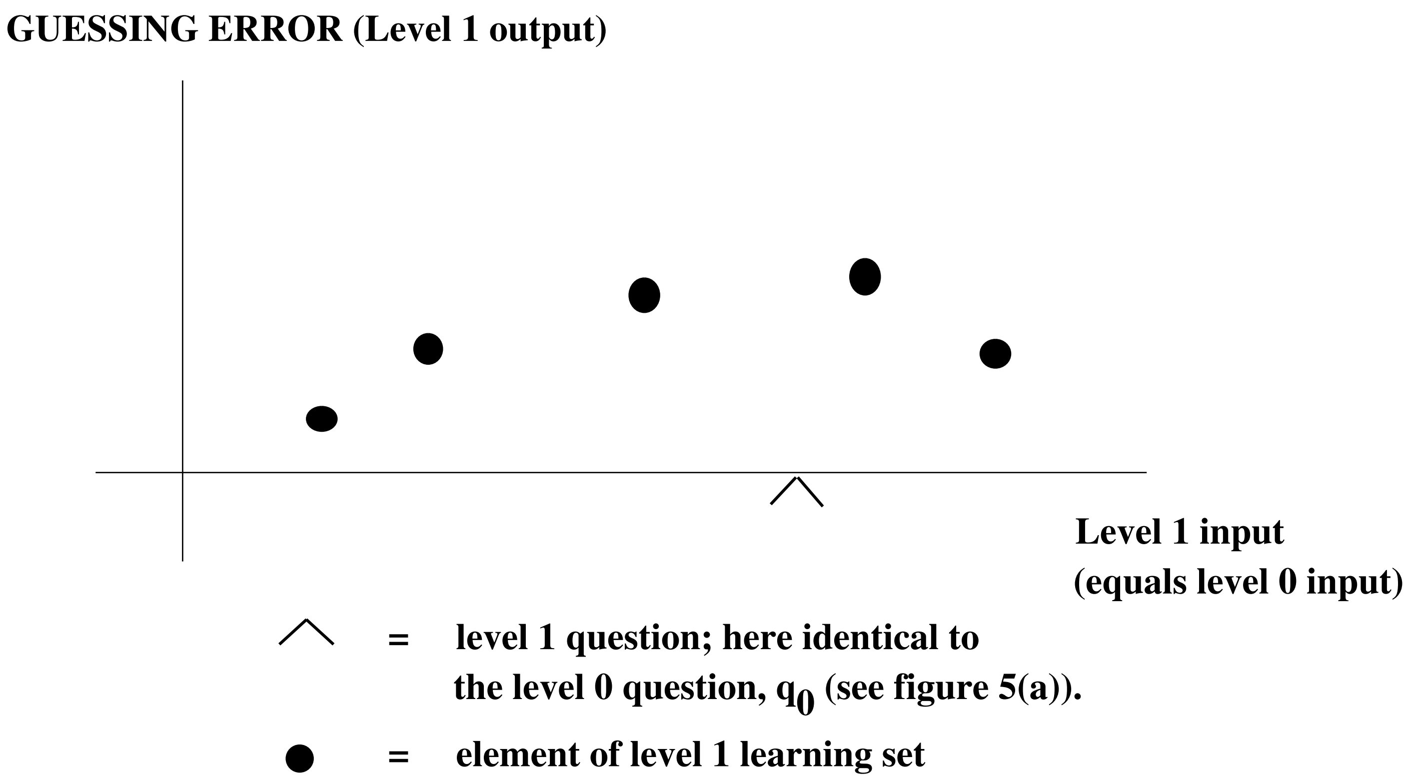

GUESSING ERROR (Level 1 output)
Level 1 input
(equals level 0 input)
= level 1 question; here identical to
the level 0 question, q0 (see figure 5(a)).
= element of level 1 learning set

Figure 5(c).

FIGURE CAPTIONS

Figure 1. An example of how to use stacked generalization to combine generalizers. Here we are combining two generalizers, G_1, and G_2. The learning set, L, is represented figuratively by the full ellipse. A question q lying outside of L is also indicated. Finally, a partition of L into two portions is also indicated; one portion consists of the single input-output pair (x, y), and the other porti9on contains the rest of L. Given this partition, we train both G_1 and G_2 on the half {L - (x, y)}. Then we ask both generalizers the question x; their guesses are g_1 and g_2. In general, since the generalizers haven't been trained with the pair (x, y), both g_1 and g_2 will differ from y. Therefore we have just learned something; when G_1 guesses g_1 and G_2 guesses g_2, the correct answer is y. This information can be cast as input-output information in a new space, i.e., as a single point with the 2-dimensional input (g_1, g_2) and the output (y). Choosing other partitions of L gives us other such points. Taken together, these points constitute a new learning set, L'. We now train G_1 and G_2 on all of L and ask them both the question q. Then we take their pair of guesses, and feed that pair as a question to a third generalizer which has been trained on L'. This third generalizer's guess is our final guess for what output corresponds to q. Assuming there's a strong correlation between the guesses made by G_1 and G_2 on the one hand, and the correct guess on the other, this implementation of stacked generalization will work well.

Figure 2. An example of how to use stacked generalization to improve a single generalizer. The (single) generalizer is G. As in figure 1, a learning set L is represented figuratively by the full ellipse, a question q lying outside of L is also indicated, and a partition of L into two portions is also shown. Given this partition, we train G on the portion {L - (x, y)}. Then we ask G the question x, and note both its guess, g and the vector from x to the nearest neighbor in {L - (x, y)}, ξ. In general,

since G hasn't been trained with the pair (x, y), g will differ from y. Therefore we have just learned something; when the question is x, and the vector from x to the nearest neighbor in the learning set is ξ, the correct answer differs from G's guess by (g - y). This information can be cast as input-output information in a new space, i.e., as a single point with the 2-dimensional input (x, ξ) and the output (g - y). Choosing other partitions of L gives us other such points. Taken together, these points constitute a new learning set, L'. We now train G on all of L and ask it the question q. Then we take the pair of q and the vector from q to its nearest neighbor in L, and feed that pair as a question to a third generalizer which has been trained on L'. This third generalizer's guess is our guess for G's error in guessing what output corresponds to q. Adding this estimated error (or a fraction thereof) back to G's guess gives our final guess. Assuming there's a strong correlation between the question and its vector to the nearest element in the learning set on the one hand, and the generalizer's error on the other, this implementation of stacked generalization will work well.

Figure 3a. A schematic depiction of a level 0 learning set and a level 0 question. Here the learning set consists of five points, indicated by solid circles. The question is indicated by a question mark (more precisely, the question is the input projection of the question mark, indicated by the intersection of the associated dotted line and the input axis.). For this example, the input space is one-dimensional.

Figure 3b. A schematic depiction of one of the pairs of the CVPS of the level 0 learning set of figure 3a. θ_{i1} consists of the four solid circles, and θ_{i2} is the fifth element of the level 0 learning set, now depicted by an open square rather than a solid circle. The other four pairs making up the CVPS simply change which element of the level 0 learning set is the square.

Figure 3c. A schematic depiction of one of the elements of a level 1 learning set. Here we determine the (two) level 1 inputs by running two level 0 generalizers on a level 0 {learning set, question} pair. For the CVPS indicated in figure 3b, both of these generalizers are taught with the solid circles, and are then asked to guess what level 0 output should correspond to the input value of the square. These two guesses form the two input components of the solid circle in *this* figure, indicated by the two dotted lines. The output of this level 1 space is the same as the output of the level 0 space, i.e., the output value of the single circle indicated in this figure is identical to the output value of the square in figure 3b.

Figure 3d. The level 1 learning set, made from the pairs of the level 0 CVPS, are indicated by the solid circles in this figure. (For clarity, only three of the five points of the level 1 learning set are shown.) Once this learning set is constructed, both level 0 generalizers are then taught with the full level 0 learning set and asked what level 0 output they think should correspond to the level 0 question. These two guesses determine the level 1 question, indicated here by (the input projection of) a question mark. A generalizer is now trained with the level 1 learning set and then makes a guess for this level 1 question. This guess serves as the full system's guess for what level 0 output should correspond to the level 0 question, given the level 0 learning set.

Figure 4. A stylized depiction of the two stages involved in the implementation of stacked generalization described in section I(iv). In the first stage the level 1 learning set L' is created from the level 0 partition set θ_{ij} and the set of level 0 generalizers $\{G_p\}$. In the second stage the exact same architecture used to create L' is used to create a level 1 question from a level 0 question. After this the final guess is found by training the level 1 generalizer on L' and then asking it the new-found level 1 question. Note that this entire procedure is twice parallelizable; once over the partitions, and once over the level 0 generalizers.

Figure 5a. Figures 5(a) through 5(c) are a geometric depiction of how stacked generalization attempts to improve the guessing of a single generalizer. The figures assume the same stacked generalization architecture as in figure 2, except that the level 1 inputs are one-dimensional, consisting solely of the level 0 input. Figure 5(a) illustrates a parent function and (part of) a learning set made up of some (noise-free) samples of that parent function. (Other elements of the learning set exist outside the range of this figure.) The level 0 generalizer is a simple connect-the-dots generalizer; its guessing is shown for the learning set is explicitly depicted. A particular question is indicated by q_0. Our task is to estimate and then correct for the error of the level 0 generalizer in guessing what output should correspond to q_0. This is achieved with a second, level 1 generalizer.

Figure 5b. See figure 5(a). To perform the stacked generalization, we need to first form the level 1 learning set. This is done via a CVPS of the original level 0 learning set. One partition pair from this CVPS is illustrated in this figure. The point in the learning set corresponding to the hatched circle is θ_{i2}; the level 0 generalizer is trained on all other points of the level 0 learning set, and then its error at guessing what output corresponds to the input component of θ_{i2} (i.e., corresponds to q_1) is tabulated. This error is the output of a point in the level 1 learning set; the corresponding level 1 input is the same as the level 0 input, q_1.

Figure 5(c). See figures 5(a) and 5(b). This figure depicts some elements of the level 1 learning set, which was made according to the algorithm described in figure 5(b). The full stacked generalization scheme works by first using a generalizer to guess what level 1 output should correspond to the level 1 question (which is identical to the level 0 question), given the input-output pairs of the level 1 learning set. After this guess is found, one finds the level 0 generalizer's guess for what output corresponds to the level 0 question, and subtracts from this level 0 guess the level 1 guess multiplied by .5 (just to be conservative). This gives our final guess for what output corresponds to the level 0 input. In this particular example, since the errors of the level 0 generalizer are so strongly correlated with the level 0 question, for any reasonable level 1 generalizer the error of the full stacked generalization scheme will be significantly lower than the error of the level 0 generalizer used straight.