HORIZONTAL GENERALIZATION

by David H. Wolpert

The Santa Fe Institute, 1660 Old Pecos Trail, Suite A, Santa Fe, NM, 87501. (dhw@santafe.edu)

Abstract: In conventional supervised learning, one searches for “vertical” patterns, connecting inputs directly to outputs. One can instead search for “horizontal” patterns, which go across the learning set, connecting one part of it with another. One way to do this is to pre-process the learning problem before feeding it to a conventional generalizer (like backpropagation). This pre-processing technique has been most fully worked out for the case where the input space is one-dimensional; in such a situation, the “pre-processing” which makes one search for horizontal patterns is simply the embedding process of non-linear time-series analysis. Although most natural for such one-dimensional problems, one can investigate how well horizontal generalization works when the input space has more than one dimension. This paper presents two sets of experiments of the efficacy of horizontal generalization for such scenarios. The first set of experiments takes place with a 24-bit input space. For a number of different target functions, a learning set is constructed by sampling the target function at 301 separate points. A version of decision-directed learning using “horizontal” pattern-searching is then used to extrapolate from that learning set to the remaining $2^{24} - 301$ points in the input space. The average error rate was 8.6%. The second set of experiments was a variation of the noisy robot arm problem recently investigated by MacKay [23]. Here again decision-directed horizontal learning is used to do extrapolation. The rms error rate was 3 times that of the noise.

Key Words: generalization and induction, learning set pre-processing, non-linear time-series analysis, extrapolation vs. interpolation.
INTRODUCTION

Consider a target function going from a subset of $\mathbb{R}^n$ to a subset of $\mathbb{R}$. This paper concerns the problem of inferring such a target function from a learning set of $m$ (perhaps noise-corrupted) samples of that function. The subset of $\mathbb{R}^n$ is the input space, and the subset of $\mathbb{R}$ is the output space. A question $q$ is any input space value. Generically, an arbitrary element of the output space will be indicated by $y$, and an element of the input space by either $x$ or $q$. An algorithm for guessing a target function from a learning set is called a generalizer. Some examples of generalizers are provided in [1-12].

Many generalizers can be viewed as algorithms which search for "vertical" patterns, mapping learning set inputs to learning set outputs. An alternative is to search for "horizontal" patterns, which instead map the learning set in one region of input space "horizontally", across to the learning set in another region of input space. In the guise of nonlinear time series analysis, generalizers which search for horizontal patterns have proven extremely successful when the input space is one-dimensional. Although some of the conventional justifications for horizontal generalization don't apply if the input space is more than one-dimensional, some do. It is therefore natural to ask how well target functions with multi-dimensional input spaces are described by horizontal patterns.

This paper investigates this issue by examining the behavior of one particular algorithm designed to search for horizontal patterns in problems with multi-dimensional input spaces. In section I, the basic idea of nonlinear time series analysis is reviewed, and an extension of it to problems with more than one dimension of input is presented. For reasons that will become clear later, this extension is called a fan generalizer (FG). Section II summarizes some computer experiments of FGs. (The reader interested in more details is directed to [22].) FGs are somewhat constrained in their real-world applicability (they are most useful when one can pick where to sample the input space to form the learning set and/or when one is interested in extrapolation rather than interpolation). Nonetheless, the results of the experiments presented in this paper suggest that some generalization problems can be fruitfully viewed in terms of horizontal rather than vertical patterns.
I. MOTIVATION AND DEFINITION OF FGs

i) Non-linear time-series analysis.

The reader interested in a detailed exposition of non-linear time-series analysis is directed to the literature (see for example [10,13-15] and references therein).

A time-series is a set of samples of a variable over time. For simplicity, we will restrict our attention to cases where the variable being sampled lives in $\mathbb{R}$. We will also restrict our attention to time-series of the form $\{y(i\tau)\}$, where $\tau$ is the “sampling period”, and $i$ ranges from 1 to some (usually large) integer $m$. The goal is to extrapolate the time-series into the future, i.e., predict what $y(k\tau)$ will be, for some $k > m$. For simplicity, we will restrict our attention to “one-step-ahead” prediction; $k = m + 1$.

The usual starting point is to assume causality. One assumes that for some number $d$, for all times $t$ the value $y(t)$ is determined by the set of $d$ numbers $\{y(t - j\tau)\}$ ($1 \leq j \leq d$). The task is to infer the dynamical law governing the evolution of the system, i.e., infer the details of how that set of $d$ numbers $\{y(t - j\tau)\}$ ($1 \leq j \leq d$) determines $y(t)$. Once the dynamics is known, we can read off the most recent $d$ points in the time-series, plug them into the inferred dynamics, and make our one-step-ahead prediction.

To infer the dynamics of the system we examine the samples of the time-series provided to us, $\{y(i\tau)\}$. These samples give us a set of $m - d$ input-output pairs of the form $\{(y(i\tau), y((i + 1)\tau), ..., y(i + (d - 1)\tau)); y(i + d\tau)\}$. Each such pair provides us with information about how $d$ consecutive samples of $y$ fix the value of the next $y$ value.

The input-output space of the $m - d$ input-output pairs is known as a “delay space” (the $d$ values 1, 2, ..., $d$ being the “delays”). The idea is to generalize in the delay space, i.e., infer a full input-output surface in the delay space from the $m - d$ samples of that surface. Such a full surface constitutes our “inferred dynamics” and allows us to make the one-step-ahead prediction.

The process of reducing the problem of one-step-ahead prediction to the problem of generalizing in the delay space is known as “embedding the time-series”. In linear time-series analysis, the generalizing is done by fitting a global hyperplane through the $m - d$ input-output pairs. If the dynamical evolution of the system is governed by the equation $y(t) = \sum_{j=1}^{d} (a_0 + a_j \times y(t - j\tau))$ for
some coefficients $a_j$, then such a fit will result in perfect prediction. (Other schemes essentially equivalent to such linear time-series analysis are linear predictive coding, autoregressive modeling, linear filtering, etc. See [16].)

There is almost always at least some non-linear aspect to any real-world dynamics. Accordingly, in non-linear time-series analysis, instead of generalizing in the delay space simply by fitting a global hyperplane, one uses a non-linear generalizer. This generalizer can either be an explicitly non-linear surface-fitter (see [10] for an example) or the non-linear generalizing can be implicit (as when back-propagated neural nets are used; see [15]).

ii) FGs as an extension of non-linear time-series analysis.

Note that we can view the problem of one-step-ahead prediction as a conventional generalization problem. The input space is time, the output space is the set of possible $y$, the set of $m$ pairs $\{(i\tau); y(i\tau)\}$ form a learning set, and the question is $(m + 1)\tau$. With conventional generalizers one searches for patterns directly connecting the input space to the output space. This means using the $m$ pairs $\{(i\tau); y(i\tau)\}$ to try to find a pattern between $t$ and $y$. For example, one might use back propagation to train a neural net to take an input layer encoding $t$ to an output layer encoding $y$.

In contrast, with non-linear time-series analysis one doesn’t directly generalize from $t$ to $y$. Rather than searching for such “vertical” patterns, instead one searches for “horizontal” patterns, connecting one part of the input-output surface $y(t)$ to another part of the surface. Embedding a time-series is simply a way to recast a generalization problem in an overtly “horizontal” form; rather than applying a generalizer directly to a problem consisting of an $m$-element learning set living in a $(1 \times 1)$-dimensional space, one first uses $d$ delays to transform it into a new “horizontal” problem, consisting of an $(m - d)$-element learning set living in a $(d \times 1)$-dimensional space. Then one does the generalizing.

The success of time-series analysis tells us that for problems with one-dimensional input spaces, horizontal patterns are often more informative than vertical ones. It is natural to investigate the degree to which this behavior also holds for problems where the dimension of the input space exceeds 1 (so that “causality” is a meaningless concept). This extension of non-linear time-series analysis to multi-dimensional input spaces constitutes FGs. The primary difference between con-
ventional non-linear time-series analysis and FGs is simply that with FGs, because the input space has more than one dimension, instead of d scalar delays \( \tau, 2\tau, \ldots, d\tau \), one has d vector delays.

To avoid the causal connotations of the word “delay”, we will from now replace it with the term “reduced”. (I.e., we will refer to the “reduced space”, the “reduced learning set”, etc., rather than the “delay space”, “the delay learning set”, etc.) The set of d vectors living in the original input space are called a “fan”. Hence the name “fan generalizer”. (See figure one.)

In addition to deciding on a value of d, in non-linear time-series analysis one must also have a means of deciding what values to assign the d delays. (In general, they need not be the successive values \( \tau, 2\tau, \ldots, d\tau \), discussed above, but can instead have gaps.) With FGs, this problem becomes how to decide on the d vectors making up the fan. In this paper we will decide amongst the possible fans by equating fans with the reduced learning sets they induce and then using cross-validation ([17-20, 8, 21]) to choose amongst (a subset of) the possible reduced learning sets. The details of this procedure, along with a discussion of some associated time-complexity issues, is presented in appendix A.

The entire process of choosing a fan, building the corresponding reduced learning set, etc., is called the fan generalizer front end (FGFE). The generalizer which works in the reduced space is called the “back-end” generalizer. As with nonlinear time series analysis, the back-end generalizer is arbitrary. In particular, one can use a neural net back end.

II THEORY AND EXPERIMENTS.

i) Theoretical advantages and disadvantages of FGs

Although the causality-based justification of nonlinear time series analysis doesn’t often apply when the input space’s dimension exceeds one, there are a number of other justifications which still hold. Perhaps the most convincing of these comes from examining closely just exactly what an FG does when it works with the simplest kind of fan, fans with a single tip. Intuitively speaking, an FG working with such fans searches for patterns of the following sort in the learning set: “Whenever I transform my input space position in such-and-such a manner, the corresponding output transforms according to such-and-such a rule.”\(^1\) One might expect many naturally-occurring target
functions to be well-described by these kinds of horizontal patterns. For example, in genetics problems, such patterns mean that when one makes a particular change to a nucleotide sequence, there is well-defined rule describing the corresponding change in the property of the sequence. (Work in progress with the splice junction problem seems to verify that such patterns do apply to some genetics problems.)

Some of the other advantages of FGs are discussed at length in [22]. Amongst them are: i) fan generalizers can combine the robustness of local generalizers with the use of the entire learning set characteristic of global generalizers; ii) so long as the reduced-space generalizer can fit a hyper-plane and there is no noise, perfect generalization accrues with FGs for target functions which are solutions to constant coefficients partial differential equation; iii) FGs directly reflect some of the symmetries of Euclidean space (e.g., neither the choice of units in the input space, nor the choice of the orientation of the input space axes, nor the choice of zeroes of those axes, etc., will have an effect on the guessing); iv) many kinds of irrelevant or misleading features in the input space are automatically filtered out by FGs, even when those “features” are not parallel to the input space axes; v) as far as FGs are concerned, the dimension of the input space is in many ways irrelevant (the back-end generalizer is never exposed to the dimensionality of the original learning set’s input space); vi) often, if a given generalizer Z is used as a back-end in an FG, the FG will achieve perfect generalization for more target functions than if Z were instead used straight on the original learning set, without any FGFE pre-processing.

In addition to such advantages, there are several disadvantages to implementing horizontal generalization with fan generalizers. First, in certain situations the time-complexity of the algorithm might grow quickly as the size of the learning set grows. There a number of ways of mitigating this problem however. (See appendix A.) A more important problem is that FGs require the (the input components of the) elements of the learning and testing sets to lie on a regular lattice. (Just as time-series analysis wants the time series to be regularly sampled in time.) Although there are ways to try to circumvent this difficulty, none currently known is entirely satisfactory. (See [22].) To circumvent it entirely it might be necessary to use some implementation of horizontal generalization other than FGs; in this paper, it is circumvented by giving the experimenter the power to decide where to sample the target function to create the learning set. Finally, even if the learning set does
lie on a lattice, if it is randomly distributed and small, and if the input space is big, there might be few fans which connect that learning set's members together more than once. (Indeed, there might be questions such that no fan connecting the question to elements of the learning set also connects elements of the learning set to each other; for such a question, there is no fan which results in a non-empty reduced learning set, and the FG can not make a guess.) Again, there are some not entirely satisfactory ways of getting around this problem, and the complete resolution of the problem might require some implementation of horizontal generalization other than FGs. In this paper, the problem is circumvented the same way the lattice problem is circumvented, by giving the experimenter the power to decide where to sample the target function to create the learning set.

At present it is not clear which of the preceding advantages and disadvantages necessarily accompany any system based on the concept of horizontal patterns, as opposed to simply accompanying that concept's implementation with FGs. Other implementations of horizontal generalization might exhibit markedly different behavior from that of FGs.

ii) Tests of FGs

In [22] thirty-six tests of the behavior of FGs encompassing nine separate problems are presented. To make things difficult, all of the tests take place with a boolean (i.e., hypercube) input space and use explicit surface-fitter generalizers as the back-end. (The back-ends used are the hyperplanar HERBIE of [7] and the weighted average HERBIE of [8]; see also [3,6,9-11,20]). Such generalizers usually perform much better when the data isn't binary-encoded; such encoding constitutes handicapping the generalizers severely.

These tests involve noise-free numerical, Boolean, and “visual” target functions. In all but one of them the FGFE improved the generalization performance, often inducing perfect generalization. The average ratio of the generalization error rate with the FGFE to the generalization error rate without the FGFE (i.e., to the error rate with the back-end used straight) was .23, +/- .05. In addition, for benchmarking purposes, standard backpropagation was also run on all thirty-six tests. In all but one of the thirty-six tests, the FG beat backpropagation. (However it should be noted that if it had been used as the back-end of an FG rather than straight, without an FGFE, backpropagation probably would have performed much better.)

For all thirty-six tests, the learning set was randomly chosen. Due to the problem discussed at the end of the preceding subsection, the randomness of the learning set meant that the size of the input space had to be limited; the hypercube had only 64 vertices. This smallness of the input space, together with the noise-free nature of the problems, means that these tests were essentially "toy" tests; they were helpful at drawing attention to various features of FGs, but had somewhat limited implications for the power (or lack thereof) of FGs as generalizers.

One situation in which the need to restrict the input space size does not occur is when one can choose where in the input space to sample the target function to create the elements of the learning set. (Such a situation also allows us to ensure that the elements of the learning set all lie on a lattice.) This is because we can choose the elements of the learning set to be contiguous - so that there isn't any difficulty in finding fans which occur more than once within the learning set - and then "accrete" guesses out from the resulting learning set: Make guesses for questions near the learning set; assume (!) that those guesses are correct and add those questions along with the associated guesses to the learning set as new input-output pairs; repeat the procedure, using the augmented "learning set" to make guesses for questions yet further removed from the original learning set, etc. With this procedure one can make a guess for any question in the input space (This "accreting" procedure is quite similar to decision-directed learning in conventional pattern recognition. See [12]. In both cases, the actual target function is never sampled after the creation of the original learning set.)

Freedom to pick where to sample the target function is not all that uncommon. It occurs, for example, in the kinds of problems traditionally addressed by expert systems; in fact, expert systems are explicitly built by having the knowledge engineer decide what questions to ask of the target function (a.k.a. the "expert"). Note also that with the accreting procedure we are explicitly using FGs to extrapolate rather than interpolate. Extrapolation is usually considered a far more difficult problem than interpolation. As such, despite the limitation of considering situations in which we are free to pick where to sample the target function, experiments with the accreting procedure constitute a good test of whether it can be fruitful to search for horizontal rather than vertical patterns.

To test the accreting procedure, eight experiments were conducted on a hypercube, and one was conducted which involved a robot arm. None of these tests were constructed with the FG in mind.
The hypercube problems had 24 bit input spaces and a single bit output space. The eight problems were:

**Tests 1-5:** The first 12 bits of an input space value are the binary encoding of a number between 0 and $2^{12} - 1$, as are the last 12 bits. The output is one of the bits of the binary encoding of the arithmetic sum of these 2 numbers. Tests 1 through 4 have the output being the first (least significant) through fourth bits of the sum. Test five has the output being the eighth bit.

**Test 6:** Here the target function has a boolean nature: if the first dimension has value 1, the output is the value of dimension 12, otherwise it's the value of dimension 24. Note that twenty-one of the input dimensions are unimportant; they're red herrings.

**Test 7:** This is the parity problem. The output is a 1 if there is an odd number of 1's in the input, 0 otherwise.

**Test 8:** The target function is the following “threshold” rule: If there exists at least one one in the first 12 bits, the output is a one; otherwise it's a zero.

For all eight of these problems a single learning set was constructed by sampling the target function at all 301 input space points containing 2 or fewer ones (out of the total of 24 bits). This learning set was then “accreted” outward, first to all questions containing exactly 3 ones, then to all questions containing exactly 4 ones, etc., all the way out to the single question containing 24 ones. This means, for example, that we are giving the FG information about the parity target function for inputs having either 0, 1, or 2 bits on, and asking it to extrapolate from that information (and only that information) to guess the parity of questions with 3 bits on, questions with 4 bits on, ..., all the way up to questions with 24 bits on.

The back-end generalizers used were the weighted average HERBIE of [8]. At each stage of the accreting procedure, the learning set consisted of the original learning set augmented by all preceding question-guess pairs answered by the FG. However for the sake of time, reduced learning sets were only constructed from the original \(< 3 \text{ bits on}\) learning set. (In other words, the “augmented learning set” was only used for setting output values at the tips of tendrils making up the fan, not for determining reduced learning sets.) For the same reason, only single-tip FGs were tested.

For the parity target function, the first-bit-of-the-sum target function, the second-bit-of-the-
sum target function, and the threshold target function, perfect generalization was achieved. The FG extrapolated from a 301-element learning set to over 16 million guesses, ranging up to a Hamming distance of 21 bits away from the original learning set, without a single mistake. The results for the other four experiments are presented in figure two. Although perfect generalization didn’t occur for these tests, the results are still encouraging. For the third-bit-of-the-sum target function, the over-all error rate was 12.5%. The “red herring” target function had the same over-all error rate. For the fourth-bit-of-the-sum target function, the error rate was 18.75%, and for the eighth-bit-of-the-sum target function it was 24.8%. It’s interesting to note that of all eight problems, the “red herring” problem, which one might intuitively expect to be the easiest for the FG, is the only one whose error rate monotonically increases with the number of ones in the questions and actually passes the random guessing value of 50%. It is also interesting to note that in three of the four cases depicted in figure two, the error frequency eventually started to drop as one moved further away from the original learning set. In other words, not only did guessing errors not exponentially propagate themselves through the accreting procedure as one might have expected (since such errors ensure that subsequent guesses will be based on an incorrect learning set), in fact the entire accreting procedure managed to be somewhat self-correcting.

Although an exhaustive analysis wasn’t possible for such a large space, the “accreting” procedure was also investigated for 100-bit input spaces. The behavior appeared to be qualitatively the same as in the 24-bit case. This isn’t too surprising, since when the original output space is binary the reduced input space and the reduced output space both consist of only two values (regardless of the number of bits in the original input space); for such situations the back-end generalizer works in a tiny input-output space.

In addition to these hypercube experiments, another experiment was conducted using the noisy two-dimensional robot arm problem recently investigated by MacKay [23] and Neal [24]. The two dimensions of input both extend from 0 to $2\pi$. As in the experiment conducted by MacKay, Gaussian noise of width .05 was added to MacKay’s robot arm target function to form a 200-element learning set. Unlike MacKay’s experiments, where the input values of the elements of the learning set was chosen at random, here the input values of the elements of the learning set were chosen to lie at the vertices of a square lattice with interval width $\pi / 10$. Note that in the units of that interval
width, the input space extends from 0 to 19 in each of the two coordinates. The 200 positions of the elements of the learning set were those in the rectangle with corner-to-corner coordinates, in units of the interval width, of \{(0, 0), (19, 9)\} (i.e., those lying on a 20 \times 10 lattice in the rectangle extending from 0 to 1.9\pi in the first input coordinate and from 0 to .9\pi in the second one). MacKay chose his (random) testing set using the same probability distribution over the input space he used to form the learning set. Accordingly, he was doing an interpolation problem rather than an extrapolation problem, where the elements of the testing set were often extremely close (in input space) to the elements of the learning set. In contrast, the experiment recounted here was one of extrapolation; the testing set consisted of 200 elements lying on the vertices of the square lattice in the rectangle \{(0, 10), (19, 19)\}.

As in the 24 bit problem, an accreting procedure was used to do the extrapolation. Questions lying in the rectangle \{(0, 10), (19, 14)\} were answered directly from the original learning set using FGs chosen so as to both minimize cross-validation error and also maximize the number of elements in the reduced learning set. (Invariably, the FGs chosen were those having two tips.) The remaining questions, those lying in the rectangle \{(0, 15), (19, 19)\}, were then answered using the 'augmented' learning set of \{the original learning set unioned with the guesses for the rectangle \{(0, 10), (19, 14)\}\}, just as in the 24 bit problems. Also just as in the 24 bit problem, the augmented learning set served to set tip values, but the reduced learning sets were not "augmented"; they were the same as in the original direct guessing for the rectangle \{(0, 10), (19, 14)\}. For simplicity, the back-end was the same as that used in linear time series analysis, namely a global hyperplane fitter.

Figure three depicts the average (over 100,000 runs) rms error on the elements of the testing set, for each of the 10 testing set lattice values in the coordinate being extrapolated (the second coordinate). Error bars on these errors are too small to show in the figure. The overall error, which is extrapolation error, is about three times the interpolation error of the systems investigated by MacKay and Neal. Note that the error actually starts to shrink when we extrapolate out more than \(\pi / 2\) radians. This is similar to the behavior of some of the 24 bit problems (see figure two). Note also that if one wanted to make a guess for a question of, say, \((7.45, 17.03)\), one could use a conventional interpolator (e.g., the version of backpropagation investigated by MacKay, or the hybrid Monte Carlo estimator of Neal) trained on the union of the original 200-elements learning set to-
gether with the guesses of the FG for the 200-element testing set. This should give much better accuracy than using the interpolator trained only on the original learning set. In this way, one can combine horizontal generalization with vertical generalization.

III. DISCUSSION

Instead of using the learning set to deduce patterns between inputs and outputs, horizontal generalizers try to deduce patterns across the learning set itself. They try to answer the question, what rules relate any one part of the learning set to another part, and can we extrapolate these rules? Horizontal generalizers try to deduce the invariants of the learning set as one moves across it.

This paper is an investigation of one particular implementation of horizontal generalization, FGs. FGs appear to be most useful when one can choose the points at which to sample the target function to create the learning set and when one is interested in extrapolation rather than interpolation. The hypercube experiments recounted in section II(ii) involved learning sets consisting of only 301 input output pairs. Yet by "accreting" guesses out from the original learning set, in four of the eight experiments perfect guessing of all 16 million-plus questions resulted, and in the remaining four experiments the average generalization error was only 17%. Similarly, the robot arm experiments recounted in section II(ii) demonstrate that in noisy situations where the input and output spaces are real-valued, FGs can make guesses almost as far from the learning set as the learning set is wide, with errors not much larger than the noise used in creating the learning set.

These results notwithstanding, no attempt has been made in this paper to conclusively prove when and how FGs will work (never mind the broader question of when and how horizontal generalization is preferable to vertical generalization). Such a complete investigation is far beyond the scope of this single paper. Indeed, it would appear that no mathematical proof of the utility of any particular implementation of horizontal generalization (as opposed to an empirical proof of it) is even possible, since generalization is an ill-posed problem. Phrased another way, one can always use knowledge of how any particular generalizer works - even a horizontal generalizer - and design a target function to defeat that generalizer.
This suggests that real-world machine learning research must satisfy itself with trying to find the defining characteristics of that subset of all machine-learning problems which humans consider "non-random", "interesting", for one reason or another, and "worth trying to learn". Loosely speaking, all machine learning can hope to do is to create a generalizer which has built into it the same biases humans use in choosing target functions. (Those biases manifest themselves either in the process by which humans make up target functions or in the process by which they pick target functions from nature.) From this perspective, the theoretical reasonableness arguments presented in this paper ("reasonable" to a human) together with the computer experiments (concerning target functions which are interesting to a human) can be viewed as a preliminary investigation suggesting that humans might have some implicit biases towards target functions based on horizontal patterns.

ACKNOWLEDGEMENTS

I would like to thank N. Rosenberg for helpful comments on the form of this manuscript. Some of this work was done under the auspices of the Department of Energy, and some of it was done at The Santa Fe Institute under the auspices of NLM grant F37 LM00011.

FOOTNOTES

[1] The input space position transformation is given by the tendril (i.e., the vector from the base of the fan to its tip), and the output transformation rule is the hypothesis function generated by training the back-end generalizer on the reduced learning set.
[2] At first, one might think that back-propagated neural nets would be far to slow to use as a back-end. However for many problems the reduced input space is tiny. For example, if the original output space is binary, and if (as was almost always the case in these tests) the fan picked has a single tip, then the back-end space has 2 possible input values and 2 possible output values. Even back-propagated neural nets can be trained extremely quickly on such a small space.

[3] These problems were chosen because they were the subject of a recent study done by a student. Some of them were previously investigated in [7] and [9].

REFERENCES


A formal definition of the fan generalizers used in this paper

This appendix defines formally the version of fan generalizers used in the experiments discussed in this paper and then briefly discusses some of the associated time-complexity issues. There are many other variations of fan generalizers. The interested reader is directed to [22].

Let $G$ be a generalizer. $G$ maps learning sets and questions to elements of the output space; $G(\text{learning set; question}) \in \mathbb{R}$. Let $\theta$ be an $m$-element learning set with elements indicated by $\{x_i, y(x_i)\}$. (Bold letters indicate vectors, and subscripts of vectors delineate one vector from another - they don't delineate components of the vector.) Let $\theta_i$ be $\theta$ with the $i$'th input-output pair removed. Then the “average cross-validation error” of $G$ on the learning set $\theta$ is defined as

$$
\sum_{i=1}^{m} \{G(\theta_i; x_i) - y(x_i)\}^2 / m
$$

(see [17-20]). It is the average error of $G$ guessing an element of $\theta$ when taught with the rest of $\theta$. (Sometimes we will refer to the average cross-validation error as the “cross-validation error” for short. The context should make it clear when such a short-hand is being used.) With this definition of cross-validation error we can now define (input-independent) FGs formally:

1) We start with an $m$-element learning set with elements indicated by $\{x_i, y(x_i)\}$; $i$ ranges from 1 to $m$. We also have a question, indicated by $q$. $q$ and the $x_i$ all live in $\mathbb{R}^n$, and the $y(x_i)$ live in $\mathbb{R}$. Finally, we have a generalizer $G$ which takes learning sets together with questions to a guess living in $\mathbb{R}$. We refer to the guess $G((x_i, y(x_i)); q)$ as using $G$ “straight”. With fan generalizers, instead of using $G$ straight we preprocess the learning set and question first with the FGFE; we “feed” $G$ via the FGFE. When being fed in this manner, $G$ is sometimes called the “back-end” of the full FG. The full FG works as follows:

2) Choose some integer $d$, where $0 < d < m$. Define an arbitrary set of vectors connecting $q$ to a subset of $d$ of the $x_i$ as the set $\{r_j\}$. $j$ ranges from 1 to $d$. Such a set of vectors is called a fan. Indicate this fan of all of the $\{r_j\}$ by $F$. An individual vector $r$ from the set $\{r_j\}$ is called a tendril.
Its base is q, and its tip is the position q + r. (See figure one.)

3) Define the learning set outputs of the tips of the fan $F$ when $F$ is lying on base $X$ as $Y_j(F, X)$, where $j$ ranges from 1 to $d$; for a particular $j$, $Y_j(F, X) = y(X + r_j)$. Depending on $X$, $F$, and the learning set, the full set $Y_j(F, X)$ may or may not exist, (i.e., depending on $X$, $F$, and the learning set, there might be a value $j$ such that the element $(X + r_j, y(X + r_j))$ does not exist in the learning set).

4) Given $F$, build a learning set of $d$-dimensional inputs and associated outputs by collecting all those pairs $\{(Y_j(F, x_i), y(x_i))\}$ for which the full set $Y_j(F, x)$ exists. (i indexes the elements of this new learning set, and $j$ now serves to index the input-space components of an element of this new learning set.) This new learning set is the reduced learning set, and $\mathbb{R}^d \times \mathbb{R}$ is the reduced input/output space. By the discussion at the end of (3), in general $i$ will only range over a subset of the integers between 1 and $m$. The cardinality of the reduced learning set is the number of such allowed $i$ values. It might be that the reduced learning set has two or more elements which share the same input components while differing in their output component. For the work presented in this paper, whenever such a case arose each of the output components of the offending elements of the reduced learning set was replaced by the average of all of those output components.

5) Calculate the cross-validation error of the reduced learning set $\{(Y_j(F, x_i), y(x_i))\}$ when fed to the generalizer $G$. (In practice, it might take too long to use leave-one-out cross-validation, the kind defined above. In this case, one should instead use some kind of leave-many-out cross-validation. See [21].)

6) Repeat step 5 for all values of $d$ and for all fans where it's possible to do so. Define $F''$ as the fan resulting in the smallest cross-validation error. (Ties are broken according to which of the fan-generated reduced learning sets has the highest number of distinct elements.)

7) The output of the FG is determined by feeding the learning set $\{(Y_j(F'', x_i), y(x_i))\}$ to the generalizer $G$ and asking of $G$ the question $\{Y_j(F'', q)\}$.

Sometimes the entire process of determining $F''$ and then feeding it along with the question $\{Y_j(F'', q)\}$ to $G$ will be referred to as the FGFE. Sometimes “FGFE” will only mean the transfor-
formation of learning set and question induced by one particular fan. The context should always make it clear how the term is being used.

Note that since in general d can grow with m, step (6) might result in a combinatoric explosion in m. This situation is not unique to fan generalizers; the exact same combinatoric explosion arises in nonlinear time series analysis as well. Note also that the combinatoric explosion isn’t as bad as might be thought at first. This is because it’s usually true that the larger the fan (i.e. the more tips it has), the less often it occurs in the learning set, and therefore the less often it has to be run with the back-end to determine its cross-validation error. Moreover, since the algorithm can be parallelized over fans, on any parallel machine the explosion can be strongly mitigated. In any case, the number of fans containing K tips is always bounded above by $C^m_K < m^K / K!$; the explosion in m is always polynomial.

Nonetheless, in practice one often wants to avoid this explosion, and it is easiest to do this by modifying step (6). For example, in practice one often restricts the set of possible fans to those for which the number of pairs in the reduced learning set exceeds some lower cut-off. One also often restricts the search to fans with d below some upper cut-off. (This strategy is often followed in nonlinear time series analysis, and was also used in the hypercube experiments recounted in section 2.)

Even given these modifications and their consequence that the number of candidate fans is small, the algorithm might still be compute-intensive (for example if the reduced training sets are large). One of way of dealing with this problem would be to use mutual information or standard correlation techniques to pick fans for which the $Y_j(F, x_i)$ are strongly correlated with the $y(x_i)$. (Indeed, some might argue that such alternative measures are preferable to cross-validation independent of any time considerations.) As another possibility, one could, for example, determine the performance measure (be it cross-validation error or something else) for all the fans which contain only a few tips and then use a greedy algorithm to search the space of larger fans. If the algorithm is still too slow, then one can simply terminate the search through fans after some time-out period, or after one finds a fan meeting some threshold value of the performance measure. This second strategy was followed in the robot arm experiment recounted in section 2.
Figures 1a and 1b. An FG. The left half of each figure represents a learning set and a question. The elements of the learning set are solid circles, the question is an open circle. There are two dimensions of input. The output value of each element of the learning set is indicated; here the outputs are all either 0, .5, or 1. Also shown in the left half of each figure is a 2-tip fan, translated over several different bases of the fan. The fan in (1a) makes a connection amongst the elements of the learning set twice. The fan in (1b) makes a connection amongst the elements of the learning set four times. Every connection defines an element in a new (reduced) learning set defined over a 2-dimensional input space. In (1a) for example the elements of the reduced learning set are ( {.5, 1}, 1) and ( {.5, 0}, 1). The reduced learning sets made in this way are indicated in the right halves of each figure. The back-end generalizer uses one of these reduced learning sets to make the FG’s guess for what output goes with the question. To decide which reduced learning set to use, we calculate the cross-validation errors of the back-end generalizer for each of the two reduced learning sets. If (for example) the reduced learning set in figure (a) has lower cross-validation error, then we answer the question by using the back-end in concert with (a)’s fan and reduced learning set. To do this, the question is converted (via the fan of (a)) to the new (reduced) question (1, .5). The back-end is then trained with the reduced learning set of (a); its guess for what output goes with the reduced question constitutes the FG’s guess for the original question.
Figure 2. The generalization performance for “accreting” guesses out from a core learning set. The input space consists of 24 bits and the output consists of 1 bit. The horizontal axis lists the number of ones in the question. The vertical axis gives the frequency of incorrect guesses over all questions with the indicated number of ones. (The number of such questions being $24! / [n! \times (24 - n)!]$, $n$ being the number of ones.) The core learning set consisted of the 301 questions having 2 or fewer ones together with the corresponding output values read off of the target function. The “accreting” of guesses proceeded from left to right; the value on the horizontal axis, minus two, gives a lower bound on the Hamming distance from the corresponding questions to the elements of the core learning set.

Circles depict the results when the target function is given by addition of the first 12 bits in the input to the second 12 bits; look at the 3rd lowest bit of this sum (the 4’s bit), that’s the value of the target function’s output.

Squares depict the results for the same target function, except that the 4th lowest bit of the sum gives the target function’s output.

Triangles depict the results for the same target function, except that the 8th lowest bit of the sum gives the target function’s output.

Crosses depict the results for the “red herring” target function. (See test 5 in section II.)
Figure 3. Extrapolation errors for the noisy robot arm problem investigated by MacKay. The input space is two-dimensional (x, y), extending from 0 to 2π in each coordinate. A 200 element learning set is made by sampling the robot arm target function on a square grid. The sampling goes from 0 to 2π in x, and from 0 to 9π / 10 in y. The sampling interval is π / 10 for both coordinates. Just as in MacKay’s experiment, the learning set here was corrupted with Gaussian noise having a standard deviation of 0.05. Given this noise-corrupted learning set, the “accreting procedure” described in the text was used to extrapolate the learning set out to a regularly spaced 200 element testing set occupying the other half of the input space. This was done 100,000 times. The overall rms error was 0.17, three times the noise. The rms error for each one of the testing set’s 10 grid values in the y coordinate is presented in the figure.