

Bounds Estimation Via Regression With Asymmetric Cost Functions

Dennis DeCoste

Monitoring and Diagnosis Technology Group
Jet Propulsion Laboratory / California Institute of Technology
4800 oak Grove 1 Drive; Pasadena, CA 91109
decoste@aig.jpl.nasa.gov

Abstract

This paper addresses a significant but mostly-neglected class of problems that we call *bounds estimation*. This includes learning empirical best-case and worst-case algorithmic complexity bounds and red-line bounds on sensor data. We propose a methodology for bounds estimation using regression with asymmetric cost functions and demonstrate its performance using a specific linear-regression method. We argue that its focus on bounds makes it preferable to existing estimation methods for bounds estimation problems.

1 Introduction

Work in the areas of artificial neural network and statistical pattern recognition over the last several years has resulted in significant progress on methods for estimation problems: learning an approximation of a generator function given a finite set of (possibly noisy and partial) patterns of the (continuous) target output and the (not necessary all continuous) inputs. In the simplest and most commonly used form, which we will call *mean estimation*, a regression network learns to predict the input-conditional mean of the target. In the most complex form, called *probability density estimation*, the network learns to predict the input-conditional probability of each possible output value, such that the sum over all of these values for each pattern is one. The popular compromise between these two extremes are *error bars* (e.g. [4], [10]), based on learned input-conditional mean plus/minus learned input-conditional variance.

In this paper, we are concerned with a large class of important problems for which we will argue that mean estimation (even with error bars) is too simplistic and that other existing probability density estimation methods are both excessively complex and somewhat inappropriate. This class includes monitoring physical systems for the sake of triggering a red alarm (e.g. NASA Space Shuttle operations), empirical determination of best-case and worst-case complexity bounds for algorithms (e.g. quicksort), and learning interval-valued evaluation func-

tions for heuristic-tree searchers such as B*[2]. We call this class of problems *bounds estimation problems*.

As far as we know, the technique which we will present in this paper is the first regression-based learning algorithm specifically designed to address bounds estimation problems. In order to better understand why existing estimation techniques are ill-suited for bounds estimation we will first discuss in some detail how they would behave on such problems. We will then present our new technique for avoiding these short-comings and demonstrate it on some simple yet illustrative examples. Finally, we identify some limitations in our current approach and suggest promising areas for future work.

2 Motivation

As initial motivation for this paper, consider the task of empirically determining accurate time complexity bounds for a given implementation of the quicksort algorithm. For a list of size N , its worst-case (but generally rare) bound is $O(N^2)$ and its best-case bound (and expected-case) is $\Omega(N \log N)$. In defining a regression network and training patterns for this example, we will pretend that we did not know these facts already. For simplification, we consider basic quicksort, without the common median-of-three enhancement. In that case, the worst-case lists are easy to describe: they are initially fully ordered.

For simplicity, assume we predefine the problem as a simple linear regression to find the six weights w_1, \dots, w_6 for the linear weighted-sum: $time = w_1 + w_2 N + w_3 N + w_4 N \log N + w_5 N^2 + w_6 2^N$. For rough estimations of the time complexity of the many computer algorithms which have a primary scaling factor N , one might hope that this specific simple linear equation would suffice. However, as we will now discuss, previous estimation methods will perform very poorly on this example, whereas our new technique finds complexity bounds that one would consider reasonable and useful.

We must stress that the only realistic goal for regression-based bounds estimation is to find bounds that satisfy some (domain-specified) acceptable rate of violation on a validation data set. Thus, "expected" bounds, even if over-tight relative to the true bounds for the entire underlying domain distribution, must be

acceptable. This is not typically a problem; for example, monitoring tasks can tolerate some small false alarm rate. In this regard, a regression-based approach must necessarily differ from a more analytic-based approach.

2.1 Mean Estimation Is Inadequate

By definition, mean estimation will, at most, be able to accurately estimate only the expected-case. Even then, trying to ensure the learning of accurate weights on the critical $N \log N$ and N^2 terms (say, to compare two quicksort implementations) would impose a great burden on us to ensure that our patterns are truly representative of the actual lists in our application domain. In particular, we would have to be especially careful not to include an unrepresentative number of patterns near either the worst-case or the best-case. For example, if we wish to randomly generate many training patterns, our generator must reflect the expected distribution of initial orderedness in our application's lists, as well as any other important factors (e.g. the amount of duplicate elements in each list, etc.). Otherwise, mean estimation will not be telling us anything of real value. For example, it already cannot confidently identify the worst-case bound, unless we actually spoonfed it only worst-case patterns (which we are assuming we do not already know).

Of course, we are also assuming the general case where both the best and worst case for many N is not known to exist in the set of training patterns. Otherwise, in this simple example of only one input, one could obviously just retain the patterns with maximum (minimum) targets for each input value, and use mean estimation to confidently find the high (low) bound. Furthermore, note that even if collecting such a complete set of patterns was possible, such input-conditional min/max target pre-filtering is not feasible in general, even for relatively small sets of inputs. It would effectively require a multi-dimensional histogram (whose size is exponential in the number of inputs times the number of bits required to represent each input value), where each bin remembers the minimum and maximum target value seen for the set of input values that bin represented.

Also note that it is not sound to even try to roughly estimate a high bound as the most complex input term with a significantly non-zero weight in a mean estimation. The reason is that estimating the mean may not require any significant accounting of the worst cases, especially if the mean case is polynomial and the (rare) worst case is exponential. Furthermore, an analogous attempt for finding low bounds is not even possible.

2.2 Error Bars Are Inadequate

Although error bars are specifically designed to reflect input-conditional confidence in mean estimations, this quicksort example illustrates that they do not adequately address the general bounds estimation problem. For example, we might indeed learn error bars within which 95% of all quicksort times for a given N will fall. But that still does not tell us how high or low the quicksort time can be for that N .

Most importantly, note that in this example, the 5% of the patterns outside of those error bars will not be normally distributed. The quicksort times lower than the low error bar will be few standard deviations away, whereas the worst-case times will likely be many standard deviations away. So, for any chosen number of standard deviations, one bar will fit better at the expense of the other. There is also the fundamental problem that input-conditional variance can be seriously underestimated for finite sets of training patterns, as discussed in [4].

Furthermore, whereas ignoring outliers is often very useful in mean estimation, it is less appropriate for bounds estimation, since such "outliers" are more likely to actually be patterns of the very bounds we are trying to learn. This is easily imagined for problems with sharp phase transitions between common polynomial and rare exponential complexity bounds (e.g. constraint-satisfaction problems [6]).

2.3 Probability Densities Are Excessive

So, to learn the worst-case and best-case bounds, we need some form of probability density estimation more powerful than error bars. One popular type is a Gaussian mixture model [3], which essentially generalizes normally-distributed error bars for multi-modal problems. However, this will not perform well for our quicksort example, since there do not exist some finite set of modes (conditional only on the sole input N) around which the quicksort times are normally distributed.

Alternatively, one could try fractional binning [10] which is a form of input-conditional histograms that performs probability density estimation while reducing the classic problem of discontinuity at bin boundaries (by averaging the outputs of neighboring bins). Due to its generality, fractional binning should eventually learn reasonable bounds, within the resolution of the fixed bin widths. But it will require simultaneously training the outputs for tens, hundreds, or perhaps even thousands of bins. Thus, we would expect its training time for bounds estimation to be orders of magnitude greater than that of a single regression. Furthermore, it would still not result in an interpretable function such as $180.1 + 1.2 \log N + 3.4 N^2$. Thus, it seems more reasonable to view fractional binning as a tractable but approximate alternative to the previously mentioned histogram-based method for partitioning patterns into high, low, and intermediate classes.

Such methods are ideal for problems for which clearly identifying the modes of multi-modal distributions is the key goal, such as learning models for many control tasks. But they are excessive for the task of bounds estimation. In our quicksort example, we are not asking to find some finite number of peaks (modes) in the non-normal distribution of quicksort time across the single input N . Nor are we asking for the input-conditional probabilities of each output value. We are simply asking what the bounds are, conditional on N .

From the perspective of the fractional binning ap-

proach, we are asking for a single bin which contains all of the targets. We do not particularly care how the probability density varies across that bin. In other words, whereas fractional binning employs some fixed number of bins with fixed widths and fixed location and learns input-conditional probabilities for each, for the problem of bounds estimation it seems more appropriate to simply fix the probability (to 1) for a single bin and learn its input-conditional width and location.

2.4 Key Problem: Insufficient Features

The underlying reason existing probability density estimation methods are excessive for bounds estimation is that they promise more than is required, and so in order to deliver on those promises they require more training patterns and more training time. Note that the factors defining the mean and probability density (e.g. for quicksort: N , degree of orderedness, degree of duplicacy, de.), can be much more complex and hard to fairly represent in a finite set of patterns than the factors defining the worst case (e.g. fully-ordered lists for several N). Thus, one of our motivating intuitions is that bounded estimation may require significantly fewer relevant features (i.e. inputs and hidden units) than probability density estimation.

If the training patterns provide all relevant inputs and our regression model has sufficient flexibility (e.g. number of hidden units), then the mean and both bounds will be within the target noise of each other. The key dilemma in practice is that a sufficient yet manageable set of features is often difficult to determine. A traditional probability density estimation framework commits to minimizing the error in estimations of the probability for each (Gaussian) bin, etc. in the regression network. So, if all of the relevant inputs are not available, this can lead to a vicious cycle (with diminishing returns) of introducing additional Gaussians, bins, etc., in an attempt to find the (possibly non-existent) right number that best matches the modality of the underlying distribution in terms of the available data. Thus, our intuition is that a more direct attempt to learn bounds instead of probability densities should be much more efficient when good features are scarce.

2.5 Hard-Limit Bounds Often Suffice

As further motivation, we stress that in our experience crisp functional limits are actually not uncommon or impractical in a variety of domains. For example, (manually-defined) high and low red-lines are pervasively used within NASA operations as thresholds for triggering alarms during automated monitoring of spacecraft for anomalies. In fact, to a very large extent, spacecraft are designed with component tolerances and sensor placements that, make such red-lining sufficient for detecting faults in many plausible future scenarios. The cost of manual formulation and validation) of red-lines and the existence of massive archives of mission sensor data together beg for automated bounds estimation. Similarly, the B* chess search algorithm [2] shows

promising use of interval-valued position evaluations, even though a more general probability distribution representation (e.g. [1]) has greater representational power (and greater computation costs). Others have successfully improved qualitative reasoning through tile fitting of piecewise-monotonic quantitative bounds [7].

Clearly, (accurate) probability density estimations (if obtainable) are more powerful than a mere pair of input-conditional bounds, particularly for an application for which a meaningful utility function can be formulated and applied across the density. That could be particularly useful for tasks such as control. But as the above examples illustrate, many analysis tasks would benefit from bounds estimation that balances accuracy against cost (both training and execution).

3 Bounds Estimation Via 2 Regressions

We define the bounds estimation problem as follows:

Definition 1 (Bounds estimation) Given \mathcal{N} patterns of target y and inputs x_i , generated from the true underlying function $y = f(x_1, x_2, \dots, x_{n-1}) + \epsilon$, find the "tightest-upper and lower bounds y_H and y_L such that $y_L \leq y \leq y_H$ holds "as much as possible" ¹ for each pattern $\langle y, x_1, x_2, \dots \rangle$, where $y_L = f_L(x_1, x_2, \dots, x_h)$ and $y_H = f_H(x_1, x_2, \dots, x_h)$.

We allow any $0 \leq h < m$ and $0 \leq h \leq m$, making explicit both our expectation that some critical inputs of the generator may be completely missing from our patterns and our expectation that some inputs may only be useful in determining one of the bounds. We also make the standard assumption that the inputs are noiseless whereas the target has Gaussian noise defined by ϵ .

3.1 Asymmetric Cost Function

Our approach is based on the simple intuition that the regression cost function should discourage outputs below (above) the target for high (low) bounds. To do this, we split the task of bounds estimation into two independent regressions over the same set of patterns one to learn the expected high bound f_H and one to learn the expected low bound f_L . We define respective asymmetric cost functions E_H and E_L across all patterns \mathcal{P} , as shown in Figure 1. ²

$$\begin{aligned}
 c_H &= \begin{cases} P_{H_u}(y_H - y)^{R_{H_u}} & \text{if } y_H \geq y \\ P_{H_o}(y_H - y)^{R_{H_o}} & \text{if } y_H < y \end{cases} & c_L &= \begin{cases} P_{L_o}(y_L - y)^{R_{L_o}} & \text{if } y_L \leq y \\ P_{L_u}(y_L - y)^{R_{L_u}} & \text{if } y_L > y \end{cases} \\
 E_H &= \sum_{p \in \mathcal{P}} c_H & E_L &= \sum_{p \in \mathcal{P}} c_L \\
 E_{|H|} &= \sum_{p \in \mathcal{P}} |c_H| & E_{|L|} &= \sum_{p \in \mathcal{P}} |c_L| \\
 \text{Parameters: } & P_{H_u}, P_{H_o}, P_{L_u}, P_{L_o} \geq 0; R_{H_u}, R_{H_o}, R_{L_u}, R_{L_o} \geq 1.
 \end{aligned}$$

Figure 1: Asymmetric cost functions.

¹We will provide parametric definitions of this and "tightness" shortly.

²For simplicity, further discussion focuses on learning f_H using cost E_H ; learning f_L using E_L is analogous.

For each pattern, P_{H_n} gives the penalty factor for an *alarm*, when the network gives an output y_H below the target y , and P_{L_n} gives the penalty for a *non-alarm*, when $y_H > y$. We will call these alarm and non-alarm patterns, respectively. During each training epoch, many patterns can jump between these two classes, denoted by the sets \mathcal{P}_a and \mathcal{P}_n , where $|\mathcal{P}_a| + |\mathcal{P}_n| = |\mathcal{P}| = \mathcal{N}$. Thus, selection of the P and R parameters should prefer symmetry in the cost function, to minimize the amount of discontinuity in the error surface.

We can favor non-alarms (i.e. looseness) over alarms (incorrectness) by making $P_{H_n} > P_{L_n}$. This is analogous to the use of nonstandard loss functions to perform risk minimization in classification tasks [3]. Note that formulating nonstandard loss functions for classification often involves relatively few decisions (e.g. how costly missing a cancer tumor is versus a false positive). In contrast, completely analogous loss functions for bounds regression would require (manually) specifying two large $O((2^n)^2)$ matrices, where n is the number of hits used to approximate the continuous targets. Our P factors are an attempt to gain benefits of using nonstandard loss without specifying that full matrix.

$P_{H_n} = P_{L_n} = 1$ and $R = R_{H_n} = R_{L_n}$ gives us the class of Minkowski-R errors [3], where $R=2$ gives standard sum-of-squares error and $R=1$ gives classic *robust estimation* (which reduces the effects of outliers). $R_{H_n} < 2$ becomes important when there are many non-alarm patterns for the same inputs for which the targets are much lower than y_H . With $R_{H_n} \geq 2$, machine precision limitations could preclude P_{H_n} from being sufficiently low to avoid those numerous non-alarm patterns from dominating the optimization result (leading to excessive alarms elsewhere).

The special symmetric case of $P_{H_n} = P_{L_n} = P_{L_n} = P_{L_n} = 1$ and $R_{H_n} = R_{L_n} = R_{L_n} = R_{L_n} = 2$ gives standard least-squares regression. Thus, a convenient property of our choice of cost function is that, in the limit of sufficient inputs and training patterns, both bounds can converge to the standard mean estimation result.

3.2 Efficient Linear Bounds Estimation

We now present a method for efficient linear bounds estimation using our asymmetric cost function E_H , for a net work consisting of a single linear output unit, of the form $y_H = \sum_{i=1}^h w_i x_i$.

Despite its simplicity, we believe that assuming linearity will be more practical for bounds estimation than is typical for other estimation problems. Generally, the expectations of the top-level task will be lower, being satisfied with avoiding alarms while not being "too loose". In contrast, other estimation-based tasks often intend to use the result, for tasks such as long-term prediction or control, in which precision is critical. Furthermore, our intention is to embed this approach in a constructive framework such as Cascade Correlation [5], for which a single linear output unit is indeed optimized by itself (upon inserting a new frozen hidden unit).

Having assumed linearity and carefully chosen our cost

function E_H , optimization via Newton's method [3] is particularly efficient: $\mathbf{w}(t+1) = \mathbf{w}(t) - \mathbf{A}^{-1}\mathbf{g}$, where \mathbf{g} is the h -row vector gradient of elements $\frac{\delta E_H}{\delta w_i}$ and \mathbf{A} is the $h \times h$ Hessian matrix of elements $\frac{\delta^2 E_H}{\delta w_i \delta w_j}$.

For our specific cost function E_H , the elements of the gradient at each epoch t can be computed by averaging over alarm and non-alarm patterns as follows:

$$\frac{\delta E_H}{\delta w_i} = \frac{1}{|\mathcal{P}_n| + |\mathcal{P}_a|} [P_{H_n} R_{H_n} \sum_{p \in \mathcal{P}_n} |y_H - y|^{R_{H_n}-1} \frac{\delta y_H}{\delta w_i} - P_{L_n} R_{L_n} \sum_{p \in \mathcal{P}_a} |y_H - y|^{R_{L_n}-1} \frac{\delta y_H}{\delta w_i}],$$

where $\frac{\delta y_H}{\delta w_i} = x_i$, since $y_H = \sum_{i=1}^h w_i x_i$.

With $c_p = |y_H - y|$ for each pattern p , the elements of

$$\mathbf{A}$$
 are partial derivatives of \mathbf{g} : $\frac{\delta E_H}{\delta w_i \delta w_j} = \frac{1}{|\mathcal{P}_n| + |\mathcal{P}_a|} [P_{H_n} R_{H_n} \sum_{p \in \mathcal{P}_n} [(R_{H_n}-2) c_p^{R_{H_n}-2} \frac{\delta y_H}{\delta w_i} \frac{\delta y_H}{\delta w_j} + Z_n c_p^{R_{H_n}-1}] + P_{L_n} R_{L_n} \sum_{p \in \mathcal{P}_a} [(R_{L_n}-2) c_p^{R_{L_n}-2} \frac{\delta y_H}{\delta w_i} \frac{\delta y_H}{\delta w_j} + Z_a c_p^{R_{L_n}-1}]$,

where $Z_n = Z_a = \frac{\delta^2 y_H}{\delta w_i \delta w_j} = 0$, since $\frac{\delta y_H}{\delta w_j} = x_j$.

For $R_{H_n} = R_{L_n} = 2$ the Hessian simplifies to:

$$\frac{\delta E_H}{\delta w_i \delta w_j} = 2P_{H_n} [\sum_{p \in \mathcal{P}_n} x_i x_j] + 2P_{L_n} [\sum_{p \in \mathcal{P}_a} x_i x_j].$$

Starting with zero weights, we batch learn with Newton until it either converges (i.e. all elements in gradient \mathbf{g} are close to zero) or exceeds a limit of 100 epochs (higher for extremely low non-alarm penalties).

We augment Newton's method with a λ parameter that gets added to the diagonals of the Hessian \mathbf{A} , providing a model trust region (as in Levenberg-Marquardt) [3]. This acts like gradient descent when Newton would increase error $E_{|H|}$ and also adds a form of weight regularization. Between epochs, we halve λ when error decreases and double it when error would have risen.

Using known methods to directly update \mathbf{A}^{-1} , one can ensure a total complexity per epoch of $O(\mathcal{N}h^2)$.

3.3 Spot-Checking Over P, R Parameters

Given our above formulations, even simple spot-checking for various reasonable values of $R_{H_n}, R_{L_n}, P_{H_n}$, and P_{L_n} provides a significant advance over the existing alternatives we have discussed. For example, in our experience, quick and reasonable results (though not optimal nor necessarily alarm-free) can often be gathered by re-running our batch Newton for each combination from value sets such as $R_{H_n} \in \{1, 2\}$, $R_{L_n} \in \{2, 4, 10\}$, $P_{H_n} \in \{1, 0.01, 1/\mathcal{N}, 0\}$, $P_{L_n} \in \{1, \mathcal{N}, \mathcal{N}^2\}$. Such spot-checking considers a wide spread of balancings between alarms and non-alarms, while obeying the reasonable constraints $R_{H_n} \geq R_{L_n}$ and $P_{H_n} \geq P_{L_n}$, to reflect our discussed strong preference to avoid alarms. The degree of difficulty in finding even rather loose low-alarm fits via such spot-checking can provide useful insights for data exploration (e.g. feature sufficiency).

We first train with the P's being 1 and the R's being 2, which is similar to mean estimation via singular value decomposition (SVD). In addition to possibly giving a sufficient result for some problems, the resulting SVD weights often prove useful for seeding training using other P and R values, as we will discuss shortly.

We currently manually evaluate the results of various P and R by comparing the number of resulting alarms and the sums of squared errors over \mathcal{P}_a and over \mathcal{P}_n . We have no formal basis for trading-off alarm count versus alarm standard error versus non-alarm standard error; ultimately this will be very task-specific.

4 Examples

For simplicity in discussing these examples, here we use the R and P parameter settings defined for the high bound when learning the low bound as well.

4.1 Complexity Bounding

As a simple demonstration of discovering quicksort-like complexity bounds, consider a mixed generator which emits two types of outputs: $y_1 = 0.2N^2 + 5$ and $y_2 = 2N \log_2 N + 3 + 2$. For the ten $N=1, 10, 20, \dots, 90$, we generate one pattern $\langle y_1, N \rangle$ and five distinct patterns $\langle y_2, N \rangle$, where the five y_2 values for each N are defined with $z = .2iN$ for $i=0, \dots, 4$. This gives a total of 60 patterns. As the first plot (2a) of Figure 2 shows, for each N all of the y_2 values cluster into short vertical bars. For $N=1$, $y_1 < y_2$, but by around $N=60$ the n^2 term begins to dominate (i.e. $y_1 > y_2$).

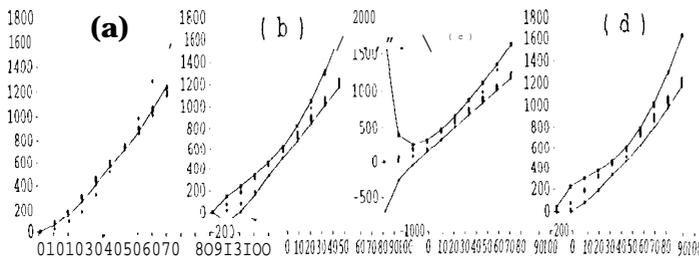


Figure 2: Example complexity Bounds

Targets plotted as dots and bounds plotted as dotted-lines.

For various P and R parameters, we trained seven linear networks, denoted by their most complex term of N : $f_1 = f(1)$, $f_{\log_2 N} = f(1, \log_2 N)$, $f_N = f(1, \log_2 N, N)$, $\dots, f_{2N} = f(1, \log_2 N, N^2, N^3, 2N)$.

The single line in plot 2a shows the mean estimation for network f_{N^2} (via SVD or similarly $R_{H_n} = R_{H_a} = 2$, $P_{H_n} = P_{H_a} = 1$). For each N the mean estimation is significantly displaced toward the more numerous targets generated from y_2 , so that variance-based error bars would not suffice to tightly bound the data.

Plot 2b shows the bounds for f_{N^2} learned using $R_{H_n} = R_{H_a} = 1$, $P_{H_n} = 0$, $P_{H_a} = 1$. The initial weights were not zero, but instead were as given by SVD. With zero weights and $P_{H_n} = 0$, the lower bound would have no error to drive it upward. SVD followed by training with $P_{H_n} = 0$ generally gives alarm-free and reasonably-tight bounds in practice, even though in theory the zero penalty would accept any alarm-free bounds that occurred during training.

In contrast, 2c shows weak bounds for f_N for the same R and P and initial weights as 2b. This illustrates a

common occurrence: networks with insufficient features tend to be readily detectable due to loose bounds. Similarly, the high bound for the wildly over-complex feature 2^N of f_{2N} was extremely loose, since training reached a weighted-term of $0.000001 \cdot 2^N$ which removed all alarms for the high bound, and $P_{H_n} = 0$ gave no error to drive learning any further.

Plot 2d shows bounds for f_{N^2} using $R_{H_n} = 2$, $R_{H_a} = 1$, $P_{H_n} = 0.01$, $P_{H_a} = 1$. The non-zero penalty P_{H_n} forced bounds inwards while the high R_{H_n} ensured that alarms were severely penalized. These are much tighter than 2b, at the price that a single very small alarm (of absolute squared error around 2) occurs in each bound. Most noticeably, the looseness in the low bound at $N=10$ for 2b has been improved. This was the best result of our small spot-checking over R and P . It yields a particularly good solution of the underlying mixed complexity of the generator: $f_{N^2 H} = 4.88 + 20.44 \log_2 N + 37.0N - 7.81N \log_2 N + .33N^2$ and $f_{N^2 L} = -1.13 - 5.4 \log_2 N - 9.1N + 3.19 \log_2 N + .02N^2$. In contrast, for same R and P : $f_{N^3 H} = 1.02 + \dots + 0.73N^2 - 0.005N^3$ and $f_{N^3 L} = 0 + \dots + 0.000021N^2 + 0.0014N^3$ suggests that the N^3 term may be overly-complex for the data.

4.2 Big-Oh Example

The closest related work is the recent work of McGeoch and Cohen [8], which presents seven heuristics for attempting to approximate the magnitude of best-case and worst-case. Their approach also requires four oracles, to provide appropriate trending and step-size determinations. They present the results of these heuristics in trying to find upper and lower bounds on the value of b of a variety of generator functions of the form $f(x) = ax^b + cx^d + e$, for $b > d$, given only six patterns for the $x = 8, 16, 32, 64, 128$ cases.

The first plot (3a) of Figure 3 shows the result of our training the previous network whose complexity is lower than the complexity of their test function for $y_1 = 1$, namely $f_{\log_2 x} = f(1, \log_2 x)$, with $R_{H_n} = R_{H_a} = 2$, $P_{H_n} = P_{H_a} = 1$. Plot (3b) shows that reusing the resulting weights to seed a training with P_{H_n} changed to 0 results in alarm-free bounds with still considerable looseness. However, getting even a loose bound for this generator is significant in that their reported that none of their heuristics were able to find either an upper or a lower bounds for 1) for y_1 .

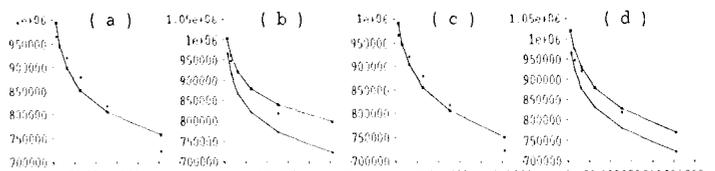


Figure 3: Hard case: $y_1 = 3x^8 - 10^4 x^6 + 10^6$.

We also learned 11 new networks of the form $f_{x^0} = f(1)$, $f_{x^1} = f(1, x^1)$, $f_{x^2} = f(1, x^1, x^2)$, \dots , $f_{x^{10}} = f(1, x^1, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9, x^{10})$. With $R_{H_n} = R_{H_a} = 2$, $P_{H_n} = P_{H_a} = 1$, the result for

$f_{2.1}$ is shown in plot (3c). With those resulting weights being reused to seed a training with P_{H_n} changed to 0, the result is plotted in 3d.

The intention of introducing these new networks was to provide a means of determining (to one significant decimal place) the power b of interest. Given that our previous complexity network f_3 is able to fit the data with little residual looseness, this would be a reasonable next step even if we didn't know the form of their generator. Manually examining the weights and errors of these networks after training with $P_{H_n}=0$ (based on initial weights from training with $P_{H_n}=1$), we noticed that even $f_{3.3}$ is able to fit the data with low error. The lowest errors (by an order of magnitude) were both achieved for the bounds of $f_{3.7}$: curiously, .7 is the power between the two competing powers of x in the generator for y_{11} . We are not claiming any sound algorithm for finding bounds b , but it is interesting to see that exploring the consequences of various R and P values can lead to potential insight about the data.

5 Discussion

This paper has argued that there exists an important but neglected class of problems called bounds estimation for which previous regression techniques can be ill-suited. We have developed a principled cost function for these problems, a specific method for efficient linear bounds estimation using this cost function, and have presented promising empirical results.

Our proposed framework supports *anytime bounding*, in which the current set of features can be used to learn bounds with no (or few) alarms, but with perhaps considerable loss. As better features become available (e.g. new sensors or external feature construction procedures), it can learn tighter bounds. We believe this anytime property will prove very useful in real-world applications such as automated monitoring. In such applications, loose bounds are useful so long as the false alarm rate is low. Loose bounds on many perspectives (e.g. raw sensors, derivatives, power spectra) can often help detect anomalies that no existing tight bounds can, since faults can manifest in so many ways.

5.1 Limitations

As mentioned earlier, if we never trained on quicksort patterns representing fully-ordered lists, we would underestimate the conceptual high bound. We view this as an example of the classic difficulty of extrapolation. We believe that using our approach to concisely remember the historically observed input-conditional bounds will be useful on its own, such as for avoiding the same false alarms during future monitoring. We currently make no strong claims about generalization ability; avoiding overfitting while adjusting R and P parameters will probably require careful use of validation sets during training.

A more fundamental limitation is that our approach is not well-suited for iterated time-series prediction, since the bounds could quickly degrade outward to extreme values. For iterated prediction, as with control, we would

expect more general (and costly) probability density estimation to usually be required.

5.2 Future Work

Although grounded on solid principles, our proposed asymmetric cost function is not the only option. Given our initial encouraging results, it seems worthwhile to also explore more exotic ones, such as ones which do not specialize to standard sum-of-squares.

It appears that one should be able to handle negative patterns (i.e. ones that *should* be above the high bound) by extending the definition of e_H to include a new penalty term for y_H being above such a negative target, with zero penalty for y_H below that target.

We also note that we have not yet tested this approach in a more general nonlinear framework, such as neural networks with hidden units. In part, however, this is because we favor first exploring pre-optimization feature construction methods (e.g. [9]), which we suspect might allow simpler linear optimization to often suffice.

For the near future, we are planning an empirical comparison on a real-world spacecraft domain against other probability density estimation techniques, to better understand when higher-precision bounds estimation might still warrant their higher complexity.

References

- [1] E. B. Baum. How a bayesian approachs a game like chess. In *Games: Planning and Learning, AAAI Fall Symposium*, 1993.
- [2] Iain J. Berliner and Chris McConnell. B* probability based search. *Artificial Intelligence*, 86(1), 1996.
- [3] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
- [4] Christopher M. Bishop and Cezhaow S. Qazaz. Regression with input-dependent noise: A bayesian treatment. *NIPS-9*, 1997.
- [5] Scott Fallman. The recurrent cascade-correlation algorithm. *NIPS-3*, 1991.
- [6] Tad Hogg, Bernardo A. Huberman, and Colin G. Williams. Phase transitions and the search problem. *Artificial Intelligence*, 81(1), 1996.
- [7] Herbert Kay and Benjamin Kuipers. Numerical behavior envelopes for qualitative models. *Proceedings of AAAI-93*, 1993.
- [8] C. C. McGeoch and P.R. Cohen. How to find big-oh in your data set (and how not to). In *Proceedings of AISTATS-97*, 1997.
- [9] Richard S. Sutton and Christopher J. Matheus. Learning polynomial functions by feature construction. In *Proceedings Of Eighth International Workshop on Machine Learning*, 1991.
- [10] Andreas S. Weigend and Ashok N. Srivastava. Predicting conditional probability distributions: A connectionist approach. *International Journal of Neural Systems*, 6, 1995.