Contents

1
Latent Variable Symbolic Regression for High-Dimensional Inputs 1
Trent McConaghy

Index 17

Chapter 1

LATENT VARIABLE SYMBOLIC REGRESSION FOR HIGH-DIMENSIONAL INPUTS

Trent McConaghy[1]
[1]*Solido Design Automation Inc., Canada*

Abstract

This paper explores symbolic regression when there are hundreds of input variables, and the variables have similar influence which means that variable pruning (*a priori*, or on-the-fly) will be ineffective. For this problem, traditional genetic programming and many other regression approaches do poorly. We develop a technique based on latent variables, nonlinear sensitivity analysis, and genetic programming designed to manage the challenge. The technique handles 340-input variable problems in minutes, with promise to scale well to even higher dimensions. The technique is successfully verified on 24 real-world circuit modeling problems.

Keywords: symbolic regression, latent variables, latent variable regression, LVR, analog, integrated circuits

1. Introduction

Symbolic regression (SR) is the automated extraction of static whitebox models that map input variables to output variables. Genetic programming (GP) (Koza, 1992) is a popular approach to do SR, with successful applications to industrial problems such as industrial processing (Kordon et al., 2005), medicine (Moore et al., 2008; Almal and al., 2006), finance (Korns, 2007; Becker et al., 2007), and robotics (Schmidt and Lipson, 2006).

In most GP-based SR applications, there are one to ten input variables, and hundreds to thousands of training samples. GP-based approaches are quite good at handling these. There are two approaches to handling more input variables. The first is to prune the variables beforehand, e.g. from neural networks (Kordon et al., 2002). The second is to let GP prune the variables on-the-fly during the SR run (Smits et al., 2005; Korns, 2007).

Pruning is reasonable when the significant variables are just a fraction of the overall set of variables. But what about when *most* variables have a degree of influence that cannot be ignored? Consider Figure 1-1 left, which is the output of a nonlinear sensitivity analysis from input/output $\boldsymbol{X}/\boldsymbol{y}$ training data. Here, the input variables are ordered from highest to lowest impact. The cumulative sum of impacts vs. variable number is plotted. While the first 10 variables explain about 50% of the total variation in y, **almost all of the variables are needed in order to capture 95% of the total variation**.

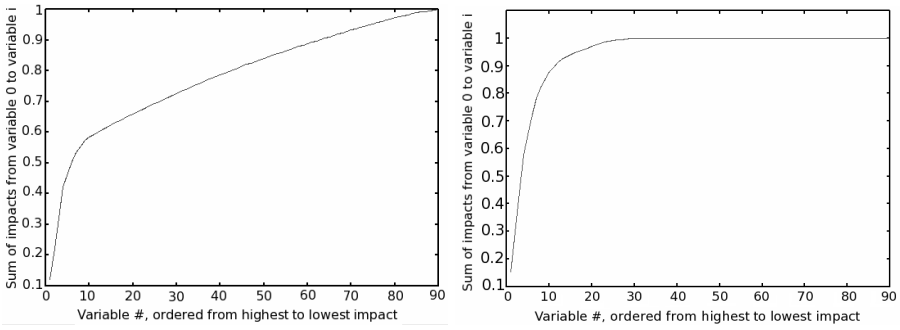

Figure 1-1. Cumulative relative impacts of input variables on a target output variable. The nonlinear impacts on the left plot were extracted using the impact-extraction technique in (McConaghy et al., 2008) where the regression models are Random Forests (Breiman, 2001). The impacts on the right plot are the weights on the model found by gradient directed regularization (Friedman and Popescu, 2004).

Because most variables are needed for a reasonable model, pruning variables will be ineffective. Even if we try a different technique that places extra bias on the most important variables (Figure 1-1 right), we still see that 1/3 of variables

– the same order of magnitude as total number of variables – are needed in order to capture 95% of the total variation.

This is the problem we face when modeling analog circuit performances as a function of manufacturing process variations. This matters, because better models allow higher-quality circuits to be designed in less time. The impact plots of Figure 1-1 were for the "AV" output of circuit in Figure 1-2 left. It has approximately 10 process variables per transistor (Drennan and McAndrew, 1999), which leads to 90 input variables overall.

In this paper, we test on 24 benchmark problems having up to **341 input variables** with impact profiles similar to Figure 1-1. Section 3 will show that a modern GP-based SR technique and several other state-of-the-art regression techniques will fail, badly, on even the easiest 16 problems. A different way to think about the symbolic regression problem is needed. So, in section 4 we introduce the perspective brought by *latent variable regression* (LVR)(Friedman and Tukey, 1974). Each "latent variable" t_i in an LVR model is a linear combination of the input variables $t_i = \boldsymbol{w_i^T x}$; and the model's output is a nonlinear function of the latent variables $\hat{f}(\boldsymbol{x}) = \sum_i g_i(\boldsymbol{w_i^T x})$. Latent variables can be thought of as auto-discovered "hidden intermediate variables" which transform the inputs into a reduced-dimensionality space. An LVR technique recently introduced in circuits (Li and Cao, 2008) is promising, but assumes a quadratic model when setting $\boldsymbol{w_i}$'s and does not return a symbolic model.

The contributions of this paper are the use of an LVR framework for solving this challenging SR problem, a means to determine the LVR linear-combination vectors $\boldsymbol{w_i}$ without assuming quadratic mapping, and, most particularly, a means to find the *symbolic* nonlinear functions g_i. We determine $\boldsymbol{w_i}$'s by building models of $\boldsymbol{x} \mapsto f$, extracting variable impacts from those models, and using those impacts as the basis for setting $\boldsymbol{w_i}$. Once $\boldsymbol{w_i}$ is determined, a (trivial for GP) one-dimensional SR run is performed having $\boldsymbol{t_i} = \boldsymbol{w_i^T x}$ as the input variable and f as the output. The process is repeated on the residuals of f until a stopping criteria is hit.

We dub our approach LVSR: Latent Variable Symbolic Regression.

This paper is organized as follows. Section 2 describes the problem setup. Section 3 gives experimental results of a modern GP technique and state-of-the-art regression techniques on the 16 benchmark problems. Section 4 introduces LVR in the context of a recent approach (Li and Cao, 2008), highlighting the promise of LVR and the current shortcomings. Section 5 introduces LVSR, which is designed to overcome the issues of past SR, regression, and LVR approaches. Section 6 has experimental validation of LVSR on 24 real-world circuit modeling problems. Section 7 concludes.

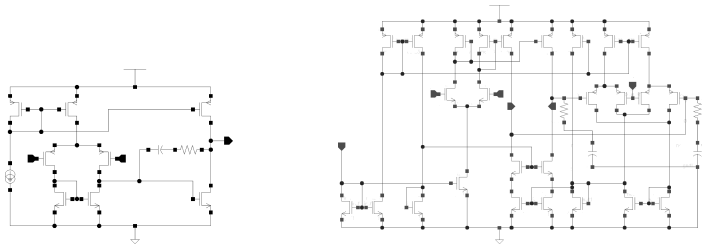

Figure 1-2. Schematics of 10-device (left) and 30-device operational amplifier (right).

2. Modeling Problems

The modeling problems come from two analog circuits as shown in Figure 1-2. These circuits are well-known to the domain experts (analog circuit designers). Each circuit's device sizes were set to have "reasonable" values by an analog circuit designer, leading to "reasonable" performance values. Each circuit has 8 performance measures of interest: AV (gain), BW (bandwidth), GBW (gain-bandwidth), GM (gain margin), OS (overshoot), PM (phase margin), SR (slew rate), ST (settling time) (Sansen, 2006).

The variations in the circuit performance due to manufacturing imprecision can be modeled as a joint probability density function (jpdf). We use the well-known model (Drennan and McAndrew, 1999) where the random variables are "process variables" which model quantities like "substrate doping concentration". Variations in these quantities affect the electrical behavior of the circuit, and therefore its performances. In this model, there are about 10 normal independent identically-distributed (NIID) random variables per transistor. In total, the 10-transistor amp had 90 random variables, and the 30-transistor amp had 215 random variables. (Section 6 will introduce an even larger problem, a 50-transistor amp with 431 input variables.)

To simulate the effect of manufacturing variations, a "Monte Carlo" (MC) analysis was performed on each circuit. In MC analysis, we draw $N = 600$ points from the jpdf. At each random point, we simulate the circuit at several sets of environmental conditions (combinations of high/low temperature, high/low power supply V_{dd}, high/low load). Each random point will get a "worst-case" value of each performance across the environmental points, which is either the minimum or maximum value (e.g. worst-case for gain "AV" is minimum value because we want to maximize gain). [1]

For our modeling problem, each random point is the model's input vector $\boldsymbol{x}$. Each worst-case performance metric is a model's scalar output, e.g. y_{AV}.

[1]The specific technology was TSMC 0.18μm CMOS. The simulator was a proprietary SPICE-like simulator of a leading analog semiconductor company, with accuracy and runtime comparable to HSPICE^{TM}.

Therefore we have 8 modeling problems with $n = 90$ input variables (for the 10T circuit), 8 modeling problems with $n = 215$ input variables (for the 30T circuit), and $N = 600$ input/output pairs per problem.

We need a scheme to assess the ability of the final models to predict on previously-unseen data. A popular approach is k-fold cross-validation, which is accurate but requires kx more computational approach than a single pass of learning. Another approach is to set aside a random subset of $\approx 25\%$ of the data for testing. This has the virtue of speed but inconsistent results, because the chosen test samples may not be representative of the whole dataset. We employ a technique which has both speed and consistency: sort the data according to the y-values, then take every 4th point for testing[1].

3. Experiments Using Traditional Regressors

This section gives results from applying a modern GP-based SR technique and several other state-of-the-art regression techniques to the problems.
We test the following regressors, which range from simple linear techniques to progressively more nonlinear approaches:

- **Least-squares (LS) linear regression**.
- **Regularized linear regression** via gradient directed regularization (**GDR**), in which a regularization term limits the variance among the linear model's weights. GDR is a generalization of both the lasso and ridge regression (Friedman and Popescu, 2004).
- **Quadratic modeling** using **PROBE**, which models the variable interactions as a rank-reduced weight matrix which improves scaling from $O(n^2)$ to $O(k * n)$ (k=rank, typically 2-10; n = number of input variables) (Li et al., 2007).
- **GP** using **CAFFEINE**, a modern SR approach which restricts the search space to interpretable-by-construction models and has demonstrated ability to scale to 100+ input variables (it *does* prune variables) (McConaghy and Gielen, 2009; McConaghy and Gielen, 2006).
- **Boosted trees** using Stochastic Gradient Boosting (**SGB**), which builds a shallow CART tree at each boosting iteration. Iterations zoom in on hard-to-model regions (Friedman, 2002).
- **Bootstrapped tree** using Random Forests (**RF**), in which each CART tree in an ensemble is greedily built from a different bootstrapped sample of the training data (Breiman, 2001; Breiman et al., 1984).

[1]This was inspired by vertical slicing (Korns, 2007) which used sorted y-values for a different purpose.

Settings for each regressor were as follows. In the notation of (Friedman and Popescu, 2004), GDR had threshold parameter $\tau = 0.2$ and stepsize $\delta\mu = 0.002$. PROBE had $max\_rank = 2$. CAFFEINE had settings like (McConaghy and Gielen, 2009), except population size of 250, population initialization size 250, and 1000 generations. SGB parameters were: learning rate α = 0.10, minimum tree depth = 2, maximum tree depth = 7, target training error = 5%. RF had 200 CARTs; CART-building would consider $\sqrt{(n)}$ input variables at each split; and splitting would continue until no possible splits remained.

Table 1-1 gives the results of the regressors on the 16 modeling problems (2 circuits x 8 problems per circuit) on the test data. Root-mean squared error $rmse(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sqrt{1/N * \sum_j^N ((\hat{y_j} - y_j)/\sigma_y)^2}$ reports the difference between y and $\hat{y}$ on testing data. Note that $rmse$ is scaled by y's standard deviation σ_y. Because SGB and RF are stochastic, for each problem we do 30 independent runs and report the median value. (We report median and not mean because the worst $rmse$ values are significantly higher, in a Poisson-like distribution.)

Table 1-1. Test RMSE values with traditional regressors. 10T = 10-transistor circuit. 30T = 30-transistor circuit. AV, BW, etc. are different circuit output metrics.

Problem	**LS-lin**	**Reg-lin (GDR)**	**Quad (PROBE)**	**GP (CAFF-EINE)**	**Boost tree (SGB)**	**Bootstr. tree (RF)**
10T AV	0.4377	0.4430	0.1384	$\gg$10.0	0.5947	0.7419
10T BW	0.6175	0.6131	0.2417	3.0170	0.7300	0.8716
10T GBW	0.4257	0.4290	0.2826	0.6016	0.5696	0.7052
10T GM	0.4404	0.4381	0.3416	0.2189	0.5524	0.6782
10T OS	0.2397	0.2506	0.2913	$\gg$10.0	0.4830	0.7002
10T PM	0.6028	0.5907	0.6710	$\gg$10.0	0.7842	0.9190
10T SR	0.0132	0.0151	0.0205	0.0555	0.4260	0.6818
10T ST	0.0566	0.0607	0.0765	$\gg$10.0	0.4379	0.6839
30T AV	0.1141	0.1158	0.1281	$\gg$10.0	0.6282	0.8118
30T BW	0.0766	0.0760	0.0949	$\gg$10.0	0.5780	0.7540
30T GBW	0.0675	0.0675	0.0766	$\gg$10.0	0.5687	0.7516
30T GM	0.1099	0.1102	0.1204	$\gg$10.0	0.6043	0.8055
30T OS	0.2165	0.2009	0.2209	$\gg$10.0	0.6101	0.7801
30T PM	0.0782	0.0844	0.1026	$\gg$10.0	0.6085	0.7665
30T SR	0.1963	0.1744	0.1903	$\gg$10.0	0.5651	0.7258
30T ST	0.1658	0.1640	0.1681	$\gg$10.0	0.6165	0.7903

Let us examine the results, one regressor at a time. As a reference, $rmse$ values of $<$0.10 are quite good, and values of $>$0.20 are very poor. The LS-linear regressor did very poorly on about half the problems, including the first

six. However, it got $rmse$ <0.10 in some problems, indicating that some of them have nearly-linear mappings. The regularized-linear regressor performed comparably to LS. The quadratic modeling approach improved upon the linear approaches for some problems, but still had very poor performance for 6/16 problems. This improved behavior that while the modeling is not quite linear and not quite quadratic, it may not be significantly more nonlinear.

The GP technique did very poorly in all but two problems. Remember that this technique did well on other 100+ variable problems. But the difference is that on those problems, pruning variables was helpful. In examining GP's behavior on the 16 problems at hand, we found that GP prunes out variables fairly aggressively, which explains its poor performance.

Both tree-based approaches did very poorly in predicting on previously-unseen inputs. There is a straightforward explanation. The quadratic models do fairly well on 10/16 problems, indicating that an assumption a continuous mapping holds fairly well. Yet the tree-based approaches, with their piecewise-discontinuous nature, do not make this continuity assumption, making the modeling problem unnecessarily difficult.

Not shown in the table, we also tested two variants of radial basis functions (RBFs) (Poggio and Girosi, 1990) (with renormalization (Hastie et al., 2001)). The first variant used Euclidian distance measure and Gaussian kernels. It gave $rmse$ values comparable to the tree-based approaches (very poor). Such performance is unsurprising, because with 100 or 200 input variables, all points are effectively far apart from all other points, rendering the Euclidian distance ineffective (Hastie et al., 2001; Smits et al., 2005). The second RBF variant used the Fractional distance measure which has been hypothesized to handle dimensionality better (Vladislavleva, 2008), but it had poor $rmse$ results too.

In summary, none of the eight "traditional" approaches tested could adequately capture the target circuit mappings. Even the best one did poorly on 6/16 problems. We need to examine the problem from a different perspective.

4. Latent Variable Regression

This section introduces latent variable regression (LVR). The general regression problem is to find a model $\hat{\boldsymbol{y}} = \hat{f}(\boldsymbol{X})$ which minimizes $rmse(\boldsymbol{y}, \hat{\boldsymbol{y}})$ on testing data $\boldsymbol{X}$. In *symbolic* regression, we also want $\hat{f}$ to be interpretable, i.e. can be inspected by a human to gain insight into the mapping.

In LVR, the mapping $\hat{f}$ is decomposed into a sum of k one-dimensional functions g_i:

$$\hat{f}(\boldsymbol{x}) = g_1(\boldsymbol{w_1^T x}) + g_2(\boldsymbol{w_1^T x}) + \ldots + g_k(\boldsymbol{w_k^T x}) \tag{1.1}$$

where each g_i takes in a scalar value $t_i = \boldsymbol{w_i^T x}$ that has been transformed from $\boldsymbol{x}$-space by projection vector $\boldsymbol{w_i}$. k is the model's *rank*; $i = 1 \ldots k$.

The power of LVR techniques is that a high-dimensional input vector $\boldsymbol{x}$ may be transformed into a one-dimensional (scalar) value t, and that nonlinear processing g is deferred until after the transformation. The LVR challenges are to find the projection vectors $\{\boldsymbol{w_i}\}\forall i$ and the nonlinear mappings $\{g_i\}\forall i$.

LVR is not new. For linear functions, it was introduced decades ago as projection pursuit (Friedman and Tukey, 1974), and related forms are called partial least squares (PLS).

The PROBE quadratic-modeling approach (Li et al., 2007) tested in section 3 can actually be interpreted as an LVR approach, where the g_i's are quadratic. Of course, the quadratic g_i's are also PROBE's weakness.

The work (Baffi et al., 1999) uses neural networks, which can handle arbitrary nonlinear mappings. However, it is slow because it iterated between finding $\boldsymbol{w_i}$'s, and finding g_i's. The approach (Malthouse et al., 1997) uses three coupled neural networks, which is complex and therefore severely prone to overfitting. The SiLVR approach of (Singhee and Rutenbar, 2007) needs just one neural network, but it only has a local optimizer for weight tuning and remains prone to overfitting. In (Jordan and Jacobs, 1994), each t_i is a neural network, and each g_i is a normalized output from an overall "gating" network. A problem with all neural-network approaches is that the g_i mapping is opaque due to the hard-to-interpret sigmoidal squashing function(s).

The recent P2M approach (Li and Cao, 2008) is of particular interest to us, because of how it decomposes the problem. In P2M, the first projection vector $\boldsymbol{w_1}$ is chosen by (1) building a PROBE model, and (2) extracting $\boldsymbol{w_1}$ from either the linear or the quadratic component of the model. Then $t_1 = \boldsymbol{w_1^T x}$ is computed for each input/output pair $j = 1 \ldots N$. Finally, an M=10-segment piecewise-linear (PWL) model of $t_1 \mapsto y$ is fit using LS, to complete the rank-1 LVR model. To build a rank-k model, the target y updates the residual $y_{target} = y_{prev} - \sum_i g_i(\boldsymbol{w_i^T x})$, and the process re-loops to the first step. P2M is particularly interesting because it demonstrated that if an algorithm could choose *good* projection vectors $\boldsymbol{w_i}$, then one could decouple learning the $\boldsymbol{w_i}$'s from the g_i's, simplifying and speeding the algorithm.

P2M has issues. First, it could choose the wrong projection vector because of the quadratic assumption, or because it must choose between quadratic vs. linear without reconciling them. Second, while the PWL model is first-order continuous, it is not second-order continuous despite experimental evidence indicating this is the case. Finally, like the neural network approaches, the PWL model is hard to interpret, which is against our *symbolic* regression goals.

With a thorough search of the GP literature, we found just one set of work using LVR (McKay et al., 1999). However, that work was tuned for low-dimensional problems (just 4 dimensions in the paper), and the output expressions were hard to interpret (e.g. $g_1 = 2.61 * exp(tanh(tanh(exp(4 * t_1)))) - 4.58$). We seek a more focused approach with more interpretable results.

5. Latent Variable *Symbolic* Regression

This section introduces latent variable symbolic regression (LVSR). Generalizing upon P2M's approach, LVSR decomposes the problem into finding projection vectors $\boldsymbol{w_i}$, finding nonlinear mappings g_i, and iterating one rank at a time. The choices within that framework are:

- To enable the *symbolic* part of latent variable symbolic regression, the g_i's are determined via GP-based symbolic regression. We use CAFFEINE (McConaghy and Gielen, 2006; McConaghy and Gielen, 2009), but any almost GP-based SR system would do here since the problem is a simple 1-d mapping.
- To choose the projection vectors, we test multiple options, each for a different reason. For a nonlinear model having discontinuities, we use Random Forests (bootstrapped trees) (Breiman, 2001) where we set each projection variable w_i, l as the impact of the l^{th} variable in the Random Forest. Its sign is computed by observing the change in y going from $\boldsymbol{x_{nominal}} = \{0, 0, ..., 0\}$ to a 1-σ perturbation in the l^{th} variable with $\boldsymbol{x} = \{0, 0, ..., 1, ..., 0\}$. We call this LVSR-RF. For a model having continuous mapping that is robust to mild nonlinearities, we use regularized linear learning with aggressive weight pruning (GDR, where $\tau = 0.95$). We call this LVSR-GDR. A bonus of using aggressive weight pruning is to reduce the final number of variables, at the possible expense of model accuracy. For completeness in comparison to P2M, we also test a quadratic model-based approach to projection-vector extraction. We call this LVSR-PROBE.

Figure 1-3 left gives the algorithm flow for LVSR.

We have also designed a further variant of LVSR, which adds tuning as shown in Figure 1-3 right. It starts by getting $\boldsymbol{w_i}$ and a *PWL*-extracted g_i. It then tunes those values, minimizing $rmse$ by changing the $\boldsymbol{w_i}$ (with n parameters) and the PWL parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ (each with$M + 1$ parameters, $M = 10$). We tune with a simple, fast, and derivative-free local optimizer (Nelder and Mead, 1965). Up to 50,000 evaluations are allowed. Each evaluation is cheap, needing just one vector-matrix product of $\boldsymbol{t} = \boldsymbol{w_i} * \boldsymbol{X}$[1], followed by simulation of the 1-d PWL model $g_i(\boldsymbol{t})$ at the N values in $\boldsymbol{t}$.

We found that, for this application, models of rank > 2 did not improve test rmse (similar to the results of (Singhee and Rutenbar, 2007)), so results shown are from max rank = 2. Runtime for all LVSR variants is on the order of a few minutes on a single-core 2 GHz CPU, with the SR portion taking the majority of time.

[1] Actually, since the optimizer changes just a subset of variables in $\boldsymbol{w_i}$, only those changes need to propagate through $\boldsymbol{X}$ to update $\boldsymbol{t}$.

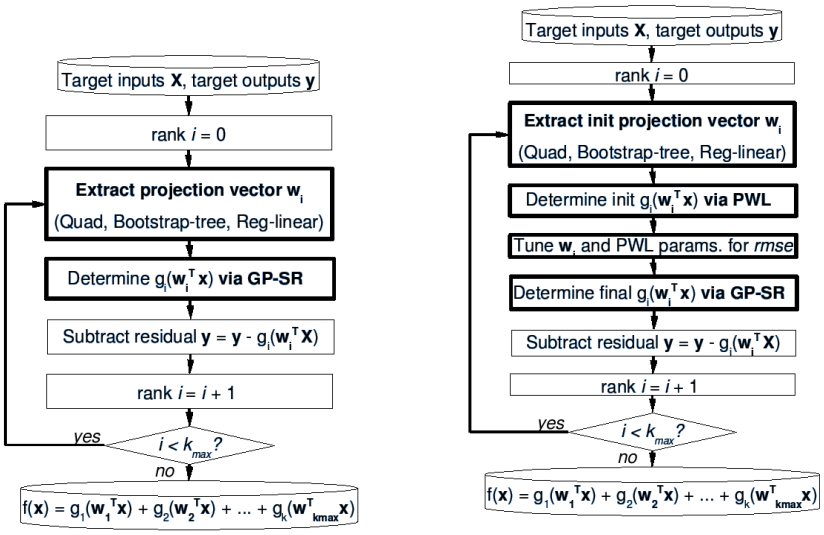

Figure 1-3. Left: Algorithm flow for Latent Variable Symbolic Regression (LVSR). The key steps are extracting the projection vector $\boldsymbol{w_i}$, and determining 1-d mapping g_i. Right: LVSR with tuning.

6. Experiments Using Latent Variable Regression

Let us first examine LVR in action, with the P2M algorithm. Figure 1-4 illustrates P2M on the 10T AV problem, where it performed the best of any regressor. The left plot shows the outcome after the first round. At any given t-value (x-axis value), the spread of points is quite tight, which indicates that the direction $\boldsymbol{w_1}$ can account for a major part of the variation. Also note that the curve on the left plot cannot be readily modeled by a linear mapping; this corresponds to the poor performance exhibited by the linear models on 10T AV seen in section 3 ($rmse$ values of 0.4377 and 0.4430). The curve can be fit fairly well by a quadratic, though not perfectly, which is why the quadratic approach PROBE did reasonably well ($rmse$ of 0.1384). On this plot, a PWL curve is able to capture the trend well, to complete the first iteration (final $rmse$ of P2M was 0.0915).

The second P2M iteration learns on the residuals of the first round. Since the first round captured most of the variation, the y-range for the second round is significantly smaller (g_2 ranges from just $\approx$-1 to $\approx$+1, whereas g_1 was from $\approx$45 to $\approx$70). The PWL model captures this as best it can, though this second round helps little. However, it illluminates a risk of PWL modeling: the model is not second-order continuous and goes to a more extreme value when extrapolating to large values of t (right hand side of the plot). This will hurt prediction ability.

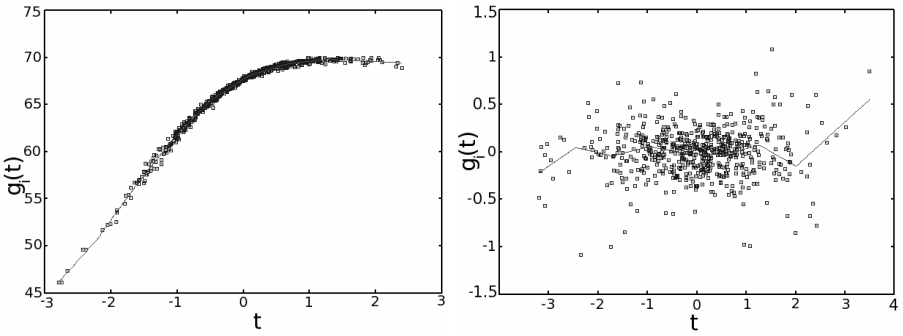

Figure 1-4. Left: Result after first round of P2M (rank=1) on 10 AV problem. The y-axis is g_1; the x-axis is the projection $t_1 = \boldsymbol{w_1^T x}$ which in this case was found via quadratic modeling (PROBE). The scatter points are the 450 training samples projected onto the g_1-t_1 plane. The line among the scatter points is a 10-segment PWL model. Right: Result second round of P2M (rank=2) on 10T AV problem, g_2 vs. t_2.

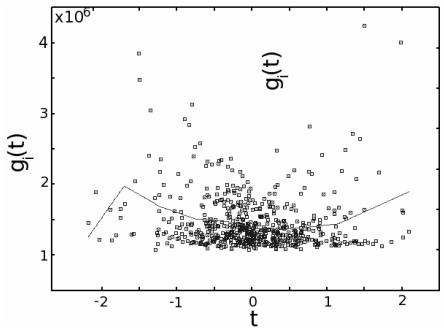

Figure 1-5. For P2M on 10T BW problem, g_1 vs. t_1.

On the next problem, 10T BW, P2M did *not* capture the direction well, as Figure 1-5 illustrates. The rank-1 projection of BW vs. $t_1 = \boldsymbol{w_1^T x}$ has a very weak pattern, with much spread in BW at any given x-axis value t. This contrast sharply with the tightly-spread rank-1 projection we just observed for AV in Figure 1-4 left. For 10T BW, the PWL model attempts to capture the weak trend, but of course results in a poor model. The rank-2 projection helps little. The final $rmse$ was 0.9077, which is the worst of any regressor.

In contrast to P2M's approach of capturing projection vectors using quadratic modeling, the LVSR approaches use impacts from either Random Forests or regularized linear learning (LVSR-RF and LVRS-GDR, respectively). Figure 1-1 is worth re-examining: it shows relative variable impacts as extracted by RF or GDR. We see that GDR needs sharply fewer variables to capture the majority of variation. This is due to the nature of the respective model-building algorithms.

RF has no bias to reduce the number of variables – given two variables causing the same effect, RF will "democratically" keep both. In contrast, GDR has bias to reduce variables – given two variables with the same effect, just one will be kept.

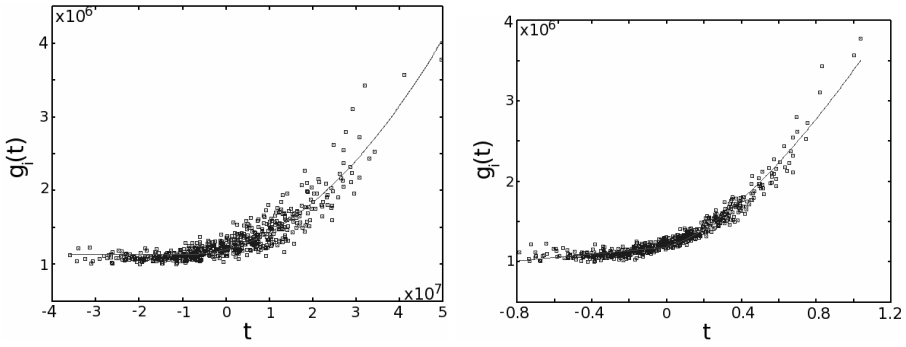

Figure 1-6. Left: For LVSR-RF on 10T BW problem, g_1 vs. t_1. Right: For LVSR-GDR on 10T BW problem, g_1 vs. t_1.

Recall that P2M did poorly on the 10T BW problem. Figure 1-6 shows the rank=1 projections from LVSR-RF (left plot), and from LVSR-GDR (right plot). Both approaches captured the trend, and GDR captured it very tightly. This is reflected in the final $rmse$ values: whereas P2M had an $rmse$ of 0.9077, LVSR-RF had $rmse$ of 0.4331E, and LVSR-GDR had $rmse$ of 0.1728 (the lowest $rmse$ for 10T BW).

Table 1-2 gives test $rmse$ values for the various LVR approaches. Because the LVSR approaches are stochastic, for each problem we do 30 independent runs and report the median value. P2M is hit and miss – sometimes it gets excellent performance but sometimes it is abysmal (e.g. $rmse$ of 0.9077). LVSR-PROBE, which uses quadratic models like P2M, performs similarly to P2M, except avoiding the abysmal failures. Because the only difference is in g_i approach, the abysmal failures are almost certainly due to the PWL mappings' poor extrapolations. LVSR-RF approach is mediocre everywhere. This is not surprising: RF tends to "soften" (lowpass filter) the variable impacts due to its "democratic" variable selection (lack of bias in choosing variables).

LVSR-GDR-tune and LVSR-GDR do the best, with comparable $rmse$ values. They both low $rmse$ in most cases, and never have abysmal performance. LVSR-GDR-tune and LVSR-GDR does better on 10T AV and 30T GM, and LVSR-GDR does better on 10T BW. So, tuning can help, but not always. There are three remaining problems that resist good models in the median (10T GBW, 10T GM, 10T PM). However, since the runs' best (minimum) $rmse$ values are 0.3797, 0.2992, and 0.3428 respectively, good models are achievable.

Table 1-2. Test RMSE values with LVR regressors

Problem	P2M: PROBE/PWL	LVSR-PROBE	LVSR-RF	LVSR-GDR	LVSR-GDR-tune
10T AV	0.0915	0.3297	0.4914	0.4012	0.1167
10T BW	0.9077	0.7018	0.6802	0.2767	0.4671
10T GBW	0.4202	0.3997	0.5271	0.4050	0.4091
10T GM	0.2723	0.3614	0.5348	0.4017	0.3738
10T OS	0.2527	0.2549	0.3807	0.2316	0.2370
10T PM	0.7188	0.5817	0.6933	0.6077	0.5937
10T SR	0.0136	0.0376	0.2913	0.0448	0.0464
10T ST	0.0574	0.0600	0.3293	0.0716	0.0556
30T AV	0.1499	0.1758	0.3744	0.1107	0.1023
30T BW	0.1058	0.1232	0.2887	0.0868	0.0777
30T GBW	0.1147	0.1038	0.3119	0.0459	0.0525
30T GM	0.1623	0.1752	0.3732	0.6198	0.1056
30T OS	0.3533	0.3393	0.3640	0.2017	0.1933
30T PM	0.1120	0.1236	0.3665	0.0856	0.0712
30T SR	0.2885	0.3749	0.3096	0.1648	0.1676
30T ST	0.2021	0.1950	0.3317	0.1673	0.1583

The final rank-1 symbolic model for one run of 10T BW, via LVSR-GDR, is given in Table 1-3. The projection vector has too many terms to interpret, but that is compensated by visualizing the g_i vs. t_i projections, the symbolic models of g_i, and if desired, a cumulative impact plot like Figure 1-1.

Table 1-3. Final model for 10T BW, as found by LVSR-GDR

$g_1(t_1) =$ 1.184e+06 + 0.871e+6 * max(0, 5.214 * t_1)$^{1/2}$ * t_1 + 0.213e+6 * t_1
t_1 = 1.338e+06 + 6.683e+03 * $DP1\_M2\_nsmm\_TOX$ + (40 other terms)

To test scalability to larger problems yet, we tested LVSR-GDR-tune on the 50-transistor circuit shown in Figure 1-7. Each modeling problem has 341 input variables. Like 10T and 30T problems, the outputs are AV, BW, etc. The rest of the setup was the same. Runtime was about the same (minutes), because the 1-d SR takes the majority of time. 30 runs were performed for each problem.

Table 1-4 gives median $rmse$ values for each of the 8 modeling problems. We see that in most cases, the $rmse$ is acceptable, and it is never abysmal. The rmse of the best run's 50T GBW was 0.2721. This signifies that LVSR has scaled very nicely to this problem with more variables; which the non-LVR approaches would have had extreme difficulty with. We expect LVSR to scale to

problems of much higher dimensionality, e.g. circuits with ≈1,000 transistors and ≈10,000 input variables. We leave that to future research.

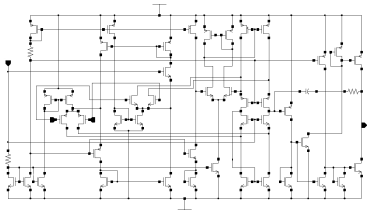

Figure 1-7. Schematic of 50-device operational amplifier.

Table 1-4. Test $rmse$ values for LVSR-GDR-tune, for 50T-amp problems having 316 input variables.

AV	BW	GBW	GM	OS	PM	SR	ST
0.2852	0.4047	0.2379	0.2265	0.2549	0.1742	0.1772	0.2162

7. Conclusion

This paper described a new challenge for GP-based symbolic regression: handling high-dimensional inputs when pruning does not work because too many variables have significant impact. This challenge matters for the real-world problem of variation-aware analog circuit design. This paper showed how how traditional GP-based SR performed poorly on such problems, alongside the poor performance of other state-of-the-art regression techniques. Then this paper introduced the latent variable regression (LVR) view of the regression problem, reviewed existing LVR techniques and their shortcomings, and introduced latent variable *symbolic* regression (LVSR). LVSR provides a symbolic model and useful visualizations of the projection vectors. On real-world circuit modeling problems, LVSR demonstrated significantly lower prediction error than traditional non-LVR approaches and a modern LVR approach (P2M).

8. Acknowledgment

Funding for the reported research results is acknowledged from Solido Design Automation Inc.

References

Almal, Arpit A. and al. (2006). Using genetic programming to classify node positive patients in bladder cancer. In *Proc. Genetic and Evolutionary Computation Conference*, pages 239–246.

Baffi, G., Martin, E.B., and Morris, A.J. (1999). Non-linear projection to latent structures revisited (the neural network pls algorithm). *Computers in Chemical Engineering*, 23(9).

Becker, Y.L., Fox, H., and Fei, P. (2007). An empirical study of multi-objective algorithms for stock ranking. In Riolo, R.L., Soule, T., and Worzel, B., editors, *Genetic Programming Theory and Practice V*, pages 241–262. Springer.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1984). *Classification and Regression Trees*. Chapman & Hall.

Drennan, P. and McAndrew, C. (1999). A comprehensive mosfet mismatch model. In *Proc. International Electron Devices Meeting*.

Friedman, J.H. (2002). Stochastic gradient boosting. *Journal of Computational Statistics & Data Analysis*, 38(4):367–378.

Friedman, J.H. and Popescu, B.E. (2004). Gradient directed regularization for linear regression and classification. Technical report, Stanford University, Department of Statistics.

Friedman, J.H. and Tukey, J.W. (1974). A projection pursuit algorithm for exploratory data analysis. *IEEE Trans. Computers*, C-23:881.

Hastie, T., Tibshirani, R., and Friedman, J.H. (2001). *The Elements of Statistical Learning*. Springer.

Jordan, Michael I. and Jacobs, Robert A. (1994). Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6:181–214.

Kordon, A., Castillo, F., Smits, G., and Kotanchek, M. (2005). Application issues of genetic programming in industry. In Yu, T., Riolo, R.L., and Worzel, B., editors, *Genetic Programming Theory and Practice III*, chapter 16, pages 241–258. Springer.

Kordon, A., Smits, G., Jordaan, E., and Rightor, E. (2002). Robust soft sensors based on integration of genetic programming, analytical neural networks, and support vector machines. In Fogel, D.B. and al., editors, *Congress on Evolutionary Computation*, pages 896–901. IEEE Press.

Korns, M.F. (2007). Large-scale, time-constrained symbolic regression-classification. In Riolo, R.L., Soule, T., and Worzel, B., editors, *Genetic Programming Theory and Practice V*, chapter 4, pages 53–68. Springer.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Li, X. and Cao, Y. (2008). Projection-based piecewise-linear response surface modeling for strongly nonlinear vlsi performance variations. In *IEEE/ACM International Symposium on Quality Electronic Design*.

Li, X., Gopalakrishnan, P., Xu, Y., and Pileggi, L. (2007). Robust analog/rf circuit design with projection-based performance modeling. *IEEE Trans. Comput.-Aided Design of Integr. Circuits and Systems*, 26(1):2–15.

Malthouse, C., Tamhane, A.C., and Mah, R.S.H. (1997). Nonlinear partial least squares. *Computers in Chemical Engineering*, 21(8).

McConaghy, T. and Gielen, G.G.E. (2006). Canonical form functions as a simple means for genetic programming to evolve human-interpretable functions. In *Proc. Genetic and Evolutionary Computation Conference*, pages 855–862.

McConaghy, T. and Gielen, G.G.E. (2009). Template-free symbolic performance modeling of analog circuits via canonical form functions and genetic programming. *IEEE Trans. Comput.-Aided Design of Integr. Circuits and Systems (to appear)*.

McConaghy, T., Palmers, P., Gielen, G.G.E., and Steyaert, M. (2008). Automated extraction of expert domain knowledge from genetic programming synthesis results. In Riolo, R.L., Soule, T., and Worzel, B., editors, *Genetic Programming Theory and Practice VI*, pages 111–125. Springer.

McKay, B., Willis, M., Searson, D., and Montague, G. (1999). Non-linear continuum regression using genetic programming. In Banzhaf, W. and al., editors, *Proc. Genetic and Evol. Comput. Conference*, pages 1106–1111.

Moore, J.H., Greene, C.S., Andrews, P.C., and White, B.C. (2008). Does complexity matter? artificial evolution, computational evolution and the genetic analysis of epistasis in common human diseases. In Riolo, R.L., Soule, T., and Worzel, B., editors, *Genetic Programming Theory and Practice VI*, Genetic and Evolutionary Computation, chapter 9, pages 125–145. Springer.

Nelder, J.A. and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, 7:308–313.

Poggio, T. and Girosi, F. (1990). Networks for approximation and learning. *Proc. of the IEEE*, 78(9):1481–1497.

Sansen, W. (2006). *Analog Design Essentials*. Springer.

Schmidt, M.D. and Lipson, H. (2006). Co-evolving fitness predictors for accelerating and reducing evaluations. In Riolo, R.L., Soule, T., and Worzel, B., editors, *Genetic Programming Theory and Practice IV*, chapter 17. Springer.

Singhee, A. and Rutenbar, R.A. (2007). Beyond low-order statistical response surfaces: Latent variable regression for efficient, highly nonlinear fitting. In *Proc. Design Automation Conference*.

Smits, G., Kordon, A., Vladislavleva, K., Jordaan, E., and Kotanchek, M. (2005). Variable selection in industrial datasets using pareto genetic programming. In Yu, T., Riolo, R.L., and Worzel, B., editors, *Genetic Programming Theory and Practice III*, volume 9 of *Genetic Programming*, pages 79–92. Springer.

Vladislavleva, E. (2008). *Model-based Problem Solving through Symbolic Regression via Pareto Genetic Programming*. PhD thesis, Tilburg University.

Index

Analog circuits, 3–4
Boosted trees, 5
Boosting, 5
Bootstrapping, 5
CAFFEINE, 5–7
CART, 5
Circuits, 3–4
Classification and regression trees, 5
GDR, 5–7, 9
Gradient directed regularization, 5–7
Gradient Directed Regularization, 9
High-dimensional inputs, 2
Integrated circuits, 3–4
Latent Variable Regression, 3, 7–14
Latent variables, 3, 7–8
Latent Variable Symbolic Regression, 3, 9–14
Linear regression, 5–8
LVR, 3, 7–14
LVSR, 3, 9–14
Manufacturing variations, 3–4
McConaghy Trent, 1
Nelder-Mead Simplex algorithm, 9
Neural network, 8
Nonlinear sensitivity analysis, 2
P2M, 8
Partial least squares, 8
Piecewise-linear model, 8–9
PLS, 8
PROBE, 5–8
Process variations, 3–4
Projection pursuit, 8
Projection vector, 7
PWL, 8–9
Quadratic modeling, 5–8
Radial basis functions, 7
Random Forests, 5–7, 9
RBF, 7
Regularized linear regression, 5–7
RF, 5–7, 9
Robust design, 3
Root mean-squared error, 6
Sensitivity analysis, 2
SGB, 5–7
Stochastic Gradient Boosting, 5–7
Symbolic regression, 2–3, 5–9
Vertical slicing, 5