Machine Learning approaches for Identifying & Characterizing Faults in Logic Simulation for Silicon Development Processes

Project Members: Mubeen Khan (mkhan3@stanford.edu); Prawal Shrastha (prawals@stanford.edu); **Project TA:** Qijia Jiang

Abstract

In the semiconductor development industry today, pre-silicon functional logic validation is one of the most intricate, engineering resource intensive and time consuming aspects of the entire silicon development process. By nature logic design validation stands out as the longest pole in the process primarily due to the following two reasons (a) time consuming nature of triaging simulation failures, and (b) lack of problem solving automation. The verification methodology as it stands today pivots on investment of engineering acumen relevant to the area of functional design along with the tedious debug process before the fault signatures can be associated with the corresponding logic in the design or the test harness. As a remedy for that problem, this study presents application of machine learning based approaches to accelerating the pre-silicon validation process.

Related Work

Research into related work[1,2,3,4,8] indicates that ML applications in this area tend to focus on post-silicon error detection predominantly. The prospective application to pre-silicon fault detection and classification outlined in this study may be somewhat unique in that aspect. Even though pre-silicon validation complexity and scope is tied closely to the microarchitecture of the device, the approaches discussed in this project should be generally applicable in concept at least.

Introduction: Device Under Test - Features & Functions

The silicon design chosen as a vehicle to demonstrate application of ML techniques to pre-si validation is a Direct Digital Frequency Synthesizer circuit (abbrev. DDFS here onwards). A brief introduction to various functional aspects of the design is necessary to gain an intuitive understanding of the behavior and associated faults for which a ML solution is being devised. The DDFS reference circuit is used in vlsi DSP applications for generating accurate & harmonically pure digital representation of high frequency time varying signals. It finds applications for generating pure Sine/Cosine wave output signal in fixed point (q1.14) format. Use cases include, but are not limited to, variable frequency tone generation for modulation/demodulation functions in high speed baseband processors such as those found in mobile wireless devices. DDFS circuits offer the following main advantages over conventional techniques (a) Extremely fast frequency switching, limited only by the speed of available logic circuits (b) Capable of very fine output resolution increasing O(2^n) with the size of the internal phase accumulator (c) Continuity of phase when frequency is changed; minimizes transients & allows to control the phase of the generated signal.

Design I/O

Inputs (Design Matrix):

i. Frequency Control (W1): Input control word "W1" determines the output frequency of the DDFS. Slow varying input
ii. Accumulator Control (W2): Moves phase accumulation forward. Fast varying input.
iii. Coarse ROM Config (s_coarse): Sine ROM coarse configuration
iv. Fine ROM Config (s_fine): Sine ROM fine configuration

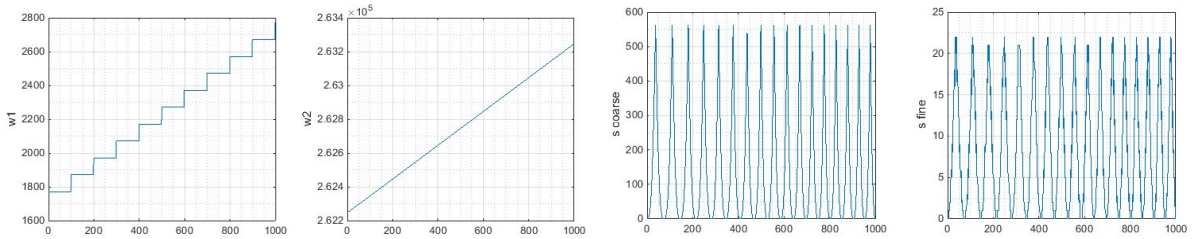

Figure 1 Input design data

Outputs:

i. Sine: 12-bit sine representation in fixed point format. Converted to normalized real time signal varying between 1 and -1

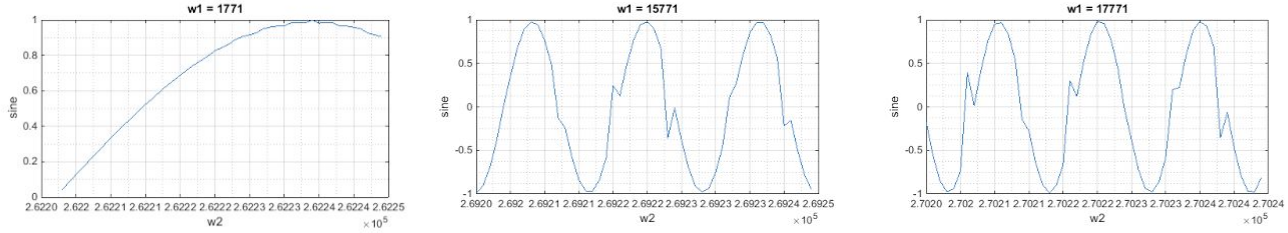

Figure 2 Synthesized sine wave output

Objectives: Circuit Behavior Modelling

Primary objective of the ML solution would be to learn the output of the circuit from the training stimulus provided and then use the learned behavior for detection of abnormal patterns that deviate from normal behavior by comparing against the hypothesized expectation. The figures below attempt to provide an intuitive understanding of the design operation along with the faults that typically manifest at the output of the circuit.

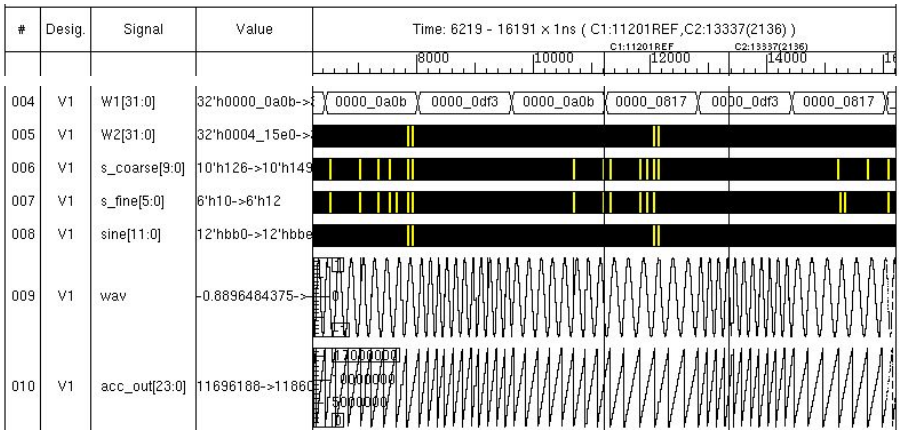

Figure 3 DDFS Design Data and Output: Frequency Modulated Sine Output

One of the challenges in this case was to identify a silicon design that was capable of furnishing data of adequate quality for both training and test especially data related to fault sources inherent to implementation, which are mainly the following:

i. Distortion due to phase truncation or accumulator overflow
ii. Distortion due to methods used to compress the data stored in the ROM
iii. Distortion due to finite precision of the data samples stored in the look-up table

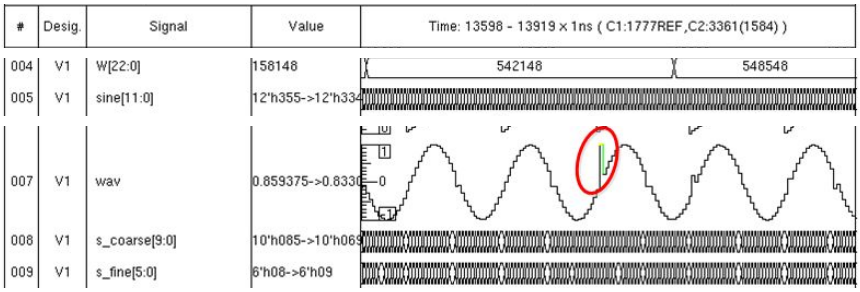

Figure 4 Phase Truncation fault due to Accumulator Overflow

Train and Test Data Set

Collection of suitable design matrix data from actual in-circuit testing that would lend itself well to a chosen ML formula was a formidable challenge. For this study such data has been collected from multiple and extensive bit-exact cycle-exact simulations of the actual design under test. The design matrix and the corresponding output for the training data set was generated from a Hardware Description Language simulation that is equivalent to in-circuit operation in every aspect. Random seeds were subsequently used to generate unseen and realistic test data sets and the fit-accuracy of the learned parameters was put to the test. All training data was carefully curated to be devoid of the defects that the estimated output from the model would be used to detect on the test data later. Multiple Train and test datasets ranging in size from 40k to 160k samples were used for the final train and error classification. The input design matrix features were processed (sphered) for zero mean and unit variance.

Models

A two step approach was adopted to address the needs of our project; (1) Development of an estimator to provide accurate prediction of design output based on input test stimulus, and (2) Formation of a classification mechanism to assimilate the output from the estimator for detection and categorization of anomalies in the test data. The solution devised therefore comprises of two models: an *output predictor* and an *error classifier*. The results obtained from the various models are enumerated in the Results section later on. Details related to model implementation are elaborated below.

Output Predictor

Several models were tried in-order to generate a predictor which are described as follows.

Polynomial Regression

At the outset, this seemed liked a reasonable approach for a predictor. A third order polynomial was chosen for the regression model. However, the results were not promising and the model was immediately abandoned.

$$h_\theta(x) = \theta_0 + \theta_1 w_1^3 + \theta_2 w_1^2 + \theta_3 w_1 + \theta_4 w_2^3 + \theta_5 w_2^2 + \theta_6 w_2 + \theta_7 s_{coarse}^3 + \theta_8 s_{coarse}^2 + \theta_9 s_{coarse} + \theta_{10} s_{fine}^3 + \theta_{11} s_{fine}^2 + \theta_{12} s_{fine}$$

K-Means++ Clustering

The k-means++ algorithm with 20 cluster centroids was used, where, instead of trying a random initialization for the cluster centers, they are spread out from each other as far as possible[6]. Interesting result here is that sine output over a continuous frequency range were grouped. Post clustering, we tried a test set and it did not do well. The phase accumulator control feature (w2) was throwing off the clustering algorithm as it has nothing to do with the frequency of the sine output. Thus we retrained the model without the w2 feature and saw better result with the test data. This is not quite the output predictor we wanted but it can be a sanity check of our output data where we verify that the output belongs to an appropriate frequency group.

LWR

LWR was anticipated to be good fit for the problem at hand but the results obtained were not found to be satisfactory.

Neural Network (NN)

NN architecture pivots on two fundamental design parameters (a) number of hidden layers, and (b) number of neurons per layer. Even though the hidden layers do not directly interact with the external environment, they have significant influence on the final output. Both the number of hidden layers and the number of neurons in each of these layers must be carefully considered. A single layer network can approximate any function that contains a continuous mapping from one finite space to another. A two layer NN can in theory represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy. As part of experimentation both architectures were evaluated for fit-performance vs. computational complexity tradeoffs. Several neuron density variations were also tried, considerations being (a) forward propagation computations (b) convergence speed (c) stability with different varying sets of training data. The neuron density that yielded the best compromise between these three conditions was preserved for dev and test.

Choice of activation function also plays an important role in the quality of learning. For all our layers we chose the tanh activation function as the output we are trying to model is real and ranges from [-1, 1]. Following is the tanh function and its first derivative.

$$f(x) = g(x) = h(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

$$\frac{d}{dx}\tanh(x) = \frac{\cosh(x)\frac{d}{dx}\sinh(x) - \sinh(x)\frac{d}{dx}\cosh(x)}{\cosh^2(x)} = \frac{\cosh^2(x) - \sinh^2(x)}{\cosh^2(x)} = 1 - \tanh^2(x)$$

Since our problem is a regression problem, we used the mean square error as our loss function; regularization was also included.

$$J = \left(\frac{1}{2m}\sum_{i=i}^{m}\left(\hat{y} - y\right)^2\right) + \lambda\left(||W^{[3]}||^2 + ||W^{[2]}||^2 + ||W^{[1]}||^2\right)$$

The delta parameters required for the three step recipe are as follows. In order to keep the report concise we have left out the gradient update expressions which are straight-forward plug and play equations.

$$\delta^{[3]} = \nabla_{z^{[3]}} J = \frac{1}{2m}\sum_{i=1}^{m}\left(\nabla_{z^{[3]}}\left(\hat{y}^{(i)} - y^{(i)}\right)^2\right) = \frac{1}{m}\sum_{i=1}^{m}\left(\left(\hat{y}^{(i)} - y^{(i)}\right) \circ \left(a^{[3](i)}\right)'\right)$$

$$\delta^{[2]} = \left(W^{[3]T}\delta^{[3]}\right) \circ \left(a^{[2]}\right)' = \left(W^{[3]T}\delta^{[3]}\right) \circ \left(1 - (a^{[2]})^2\right)$$

$$\delta^{[1]} = \left(W^{[2]T}\delta^{[2]}\right) \circ \left(a^{[1]}\right)' = \left(W^{[2]T}\delta^{[2]}\right) \circ \left(1 - (a^{[1]})^2\right)$$

L2 regularization was used to mitigate overfitting which is an undesirable side effect of NN based models. It was also ensured that the training data sets used were as defect free as possible so that any behavioral deviation in test data would be detectable from the difference of the estimated and actual outputs. In order to make the NN work properly we used 5 previous samples as part of our input features. Experiments were run to decide between 2/1 hidden layers which are discussed later. The NN based model provided the most optimal results wr.t. output and error prediction and emerged as the natural choice for the DDFS estimator.

Anomaly Detection and Classification

Error classification for our DUT pivots on Median Absolute Deviation that is derived from the estimated and actual test outputs. In the preliminary implementation devised for this particular circuit, classification was setup for detection of two signatures that may manifest in the output due to flaws or limitations in the design, Phase Truncation and Finite Precision. A Softmax regression with

Cross Entropy loss minimization was setup to classify the output samples into three categories (1) Good, or (2) Phase truncated, or (3) Finite Precision limited. Training data for the Softmax model had to be curated to include the classification labels for the output in order to determine the loss/accuracy of the model predictions. The initial accuracy numbers obtained were encouraging but not quite deployable, the procedure is still under refinement at the time of this writing. The classification model outputs probabilities denoting the likelihood of the output symbol.

$$h_{\theta_1,\ldots,\theta_k}(X \in R^n) = \begin{Bmatrix} \phi_1 \\ . \\ . \\ . \\ \phi_k \end{Bmatrix} \qquad \forall\ i = 1,\ldots,k; \quad \phi_i = \frac{e^{\theta_i^T x}}{\sum_{j=1}^{k} e^{\theta_j^T x}}; \quad where\ \phi_i = P(y = i|x)$$

For our model ϕ_1 denotes the probability of good output sample, ϕ_2 denotes the probability of a sample with Phase Truncation, and ϕ_2 denotes the probability of a sample with a Finite Precision error.

Results

K-Means++ Clustering

Figure on the left shows that the cluster centers stay constant for a continuous range of w1 values. Thus, continuous block of frequencies are grouped together and so the model can potentially be used for output frequency verification. Figure on the right shows convergence roughly after 5 iterations. Since our focus was on output prediction we did not further pursue this model.

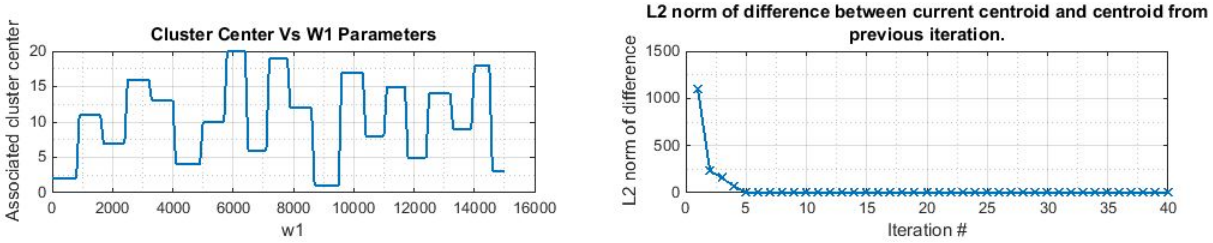

Figure 5 Left: Clustering of w1 parameters. Right: Change in centroid over time

Neural Network

A one hidden layer and two hidden layer neural networks were implemented for the predictor model. After few rounds of trial and error, we were able to tune into design parameters that worked well. Following table summarizes the final settings we ran with. The K1 parameter is the number of nodes in the first hidden layer and K2 is the number of nodes in the second hidden layer.

Model	K1	K2	Learn Rate	λ	Iterations	Train Loss	Dev Loss	Test Loss
One Hidden Layer	8	N/A	1	0.0015	250	0.0234	0.0211	0.0195
Two Hidden Layers	16	16	1	0.005	250	0.0282	0.0242	0.0235

From the table above you can see the training loss was lower for the single hidden layer model. Also, because of one less layer and fewer nodes, this model is faster, so we stayed with it. One detail to note on the table is that the development loss and test loss were much lower than the training loss. This is an unconventional result. We think the reason behind this is that we train with a data set that has a lot more frequency variation in the output than the dev and test set. Thus, the training model is inherently prone to more error. One way to verify would be to generate a new set of data with equal number of variation in frequency as the train set. However, due to lack of time we were not able to do so. Figures below demonstrate that we were able to predict the outputs fairly well. There is some attenuation in our prediction and is an aspect we would like to pursue if we had more time. The single hidden layer NN converges in less than 50 iterations whereas the two hidden layer NN converges in around 100 iterations. Thus going with the single hidden layer model further reduces the training time. Looking at the actual vs prediction output for the test set, the predictor tracks frequency modulation quite well.

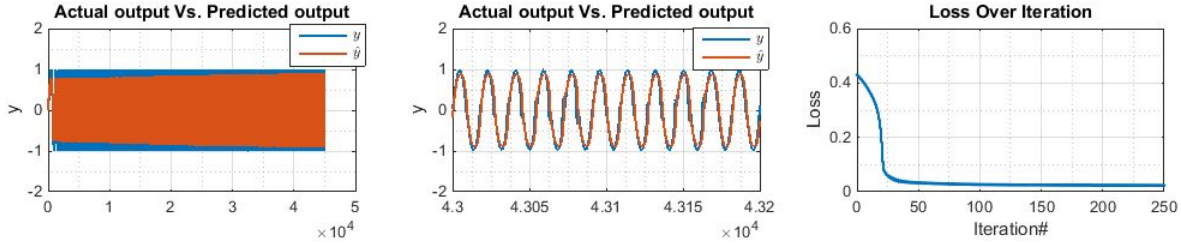

Figure 6 Training result for single layer NN.

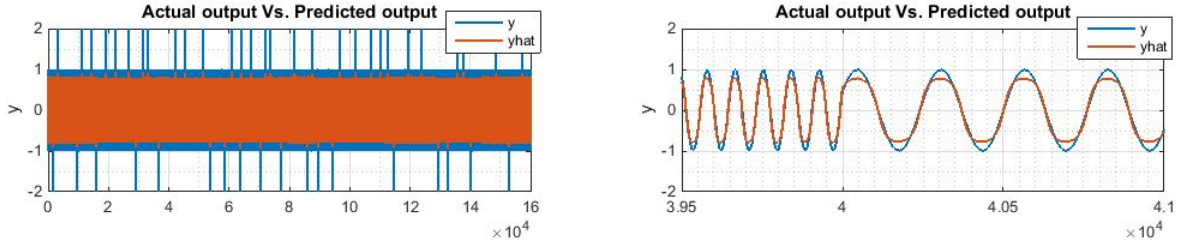

Figure 7 Test result for single layer NN.

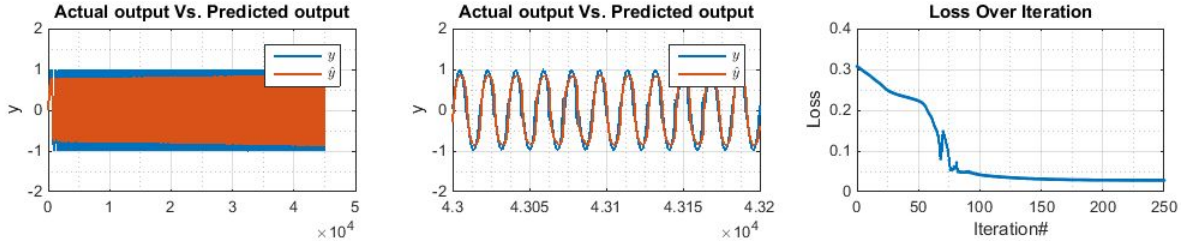

Figure 8 Training result for two layer NN.

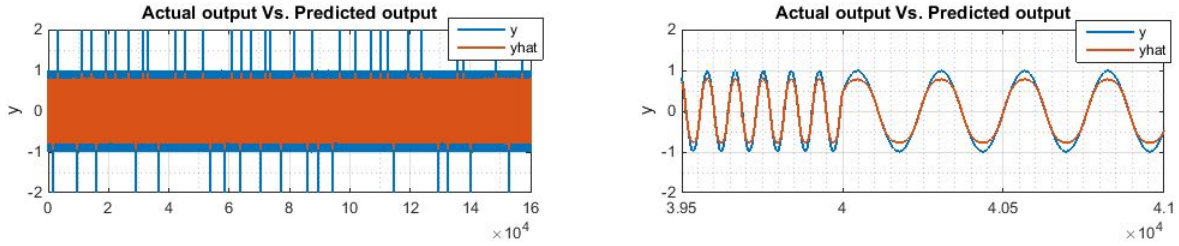

Figure 9 Test result for two layer NN.

Anomaly Detection

Error classification symbols were created by thresholding the median absolute deviation to different levels. Phase truncation manifests with a large voltage swing in the output signal. Finite Precision manifests as a step in the phase transition of the voltage, which is otherwise smoothe. The median absolute deviation thus shows two clear thresholds which are then used for creating the training symbols.

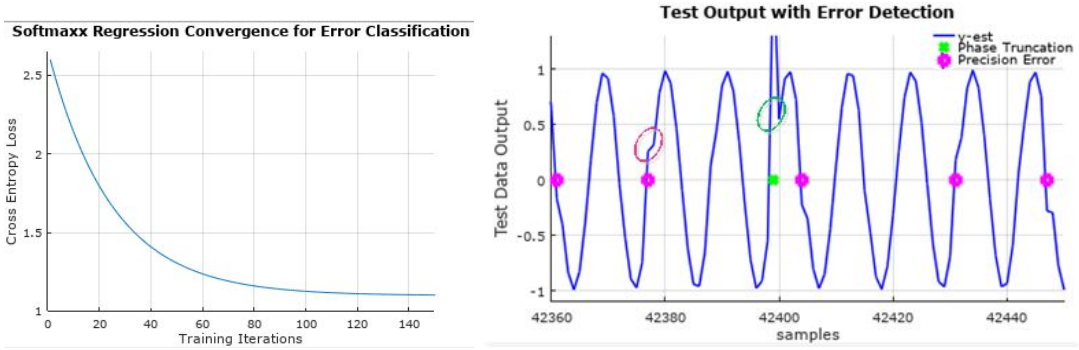

Figure 10 Anomaly Classification using Softmax Regression

Conclusion

The two step error classification approach for our simulation based validation problem has yielded some promising results and seems like a step in the right direction. This particular circuit was chosen as the test vehicle because the circuit functionality is intuitively understandable from a design I/O perspective. Application of the same technique to some other design with more obtuse I/O would have required different visualization to convey the same information. The method however lends itself well to general application to simulation based validation. Lack of prior art in ML applied to pre-silicon verification tends to indicate the uniqueness of this problem area. The project itself was a great learning experience into solidifying our understanding of supervised machine learning. The problem seemed quite formidable at the onset, the solution however materialized piece by piece, and yielded some useful results towards the end. We would like to acknowledge the valuable insights from our project advisor early on that directed us towards a meaningful solution. It has also bolstered our confidence in applying the learnings to our future projects that we may take on.

References

[1] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3), 2009.
[2] A. DeOrio, D. S. Khudia, and V. Bertacco. Post-silicon bug diagnosis with inconsistent executions. In *Proc. ICCAD*, 2011.
[3] O. Guzey, L.-C. Wang, J. R. Levitt, , and H. Foster. Increasing the efficiency of simulation-based functional
[4] D. Josephson. The manic depression of microprocessor debug. In *Proc. ITC*, 2002.
[5] S.-B. Park, A. Bracy, H. Wang, and S. Mitra. BLoG: Post-silicon bug localization in processors using bug localization graphs. In *Proc. DAC*, 2010.
[6] D. Arthur and S. Vassilvitskii. K-means++ The Advantages of Careful Seeding.
[7] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median.
[8] Andrew DeOrio, Qingkun Li, Matthew Burgess, Valeria Bertacco, Machine learning-based anomaly detection for post-silicon bug diagnosis. Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013
[9] AI FAQ/neural Nets. http://www.faqs.org/faqs/ai-faq/neural-nets
[10] Neural Network Design. http://hagan.okstate.edu/NNDesign.pdf
[11] Q.V. Le, A Tutorial on Deep Learning. Autoencoders, Convolutional Neural Networks and Recurrent Neural Networks
[12] Averill M. Law, Simulation Modelling and Analysis
[13] AC MCCORMICK, AK NANDI., Fault detection using support vector machines and artificial neural networks, augmented by genetic algorithms
[14] YL Murphey, MA Masrur, ZH Chen, Model-based fault diagnosis in electric drives using machine learning. IEEE/ASME Transactions on Mechatronics (Volume: 11, Issue: 3, June 2006)

Project Code

Our project code resides at the following google drive location. There is a project code description document that describes how the code is organized.
https://drive.google.com/drive/folders/1S3OQi-5dN3vYpzA1YMgwIEvpkEjHxsiQ

Contributions

As a team we would like to state that this was a complete joint effort and a triumph of long distance collaboration, both of us being remote SCPD students located on opposite sides of the continent.

1. Design under test simulation for harvesting train and test data sets. Many different data sets were generated for both training and test to check for model sensitivity to variations in design matrix data. (mkhan3)

2. Modeling
    a. K-Means Clustering (prawals)
    b. Polynomial Regression (prawals)
    c. LWR (mkhan3)
    d. Neural Network
        i. Single Layer (prawals/mkhan3)
        ii. Dual Layer (prawals/mkhan3)
    e. Maximum Absolute Deviation (prawals/mkhan3)
    f. Softmax Classification for Error detection (mkhan3)

3. Documentation (prawals/mkhan3)
    a. Proposal
    b. Milestone
    c. Poster
    d. Final