Application of Machine Learning to predict Boron Tube Diffusion Process parameters for optimum Efficiency

Undergraduate Thesis

Submitted in partial fulfillment of the requirements of
BITS F421, Thesis

by

Ananya Kshatrapati Shivaditya
2013B1A70792G

Under the supervision of
Dr. Rolf Stangl
Solar Energy Research Institute of Singapore, National University of Singapore

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
K. K. BIRLA GOA CAMPUS

December 2017

Acknowledgements

Thank you Dr. Rolf Stangl, for your continuous support.

Thank you Dr. Bharath Deshapnde for your guidance.

Thank you Gautam Anand and Rahul Jaiswal.

Sincere gratitude to my university, Birla Institute of Technology and Science, K. K. Birla Goa Campus, for making this thesis possible.

To the professors, staff members and students at Solar Energy Research Institute of Singapore, thank you all for your assistance and well wishes.

Certificate

This is to certify that the thesis entitled, "*Application of Machine Learning to predict Boron Tube Diffusion Process parameters for optimum Efficiency*" and submitted by Ananya Kshatrapati Shivaditya ID No. 2013B1A70792G in partial fulfillment of the requirements of BITS F421 Thesis embodies the work done by her under my supervision.

Dr. Rolf Stangl

Supervisor
Dr. Rolf Stangl
Head, Novel Cell Concepts & Simulation Group,
Solar Energy Research Institute of Singapore, National University of Singapore
Date: 11. Dec. 2017

On Campus Supervisor
Dr. Bharat Deshpande
Head of Department, Department of Computer Science,
Birla Institute of Technology and Science, Pilani - K. K. Birla Goa Campus
Date:

Abstract

Application of Machine Learning to predict Boron Tube Diffusion Process parameters for optimum Efficiency

Ananya Kshatrapati Shivaditya, 2013B1A70792G
under the supervision of Dr. Rolf Stangl
under the co-supervision of Dr. Bharath Deshpande

First Semester 2017-2018
Birla Institute of Technology and Science

In the solar industry, there is a struggle to climb up the slope of efficiency in solar modules. Clean rooms strive to alter the parameters in the processes of manufacture to get better solar modules, with a larger portion of energy from sunlight converted to electricity via photovoltaics. We explore ways to apply machine learning, and optimize the output of the Boron Tube Diffusion Process in building a solar cell precursor. In the first problem statement, we use Random Forest Regression to predict six of the process parameters from the surface concentration and depth of the diffusion profile they'd achieve. In the second problem statement, we use a 3 layer neural network to predict a new set of six process parameters, along with the value of higher implied efficiency they'd achieve. This thesis is aimed at providing proof of concept in use of machine learning for optimization inside clean room, in order to accelerate optimization in terms of time and accuracy.

Contents

Acknowledgements i

Certificate ii

Abstract iii

Contents iv

1 Introduction 1
- 1.1 Need for optimization in Solar Energy Research . . . 1
- 1.2 Boron Tube Diffusion Process . . . 1
  - 1.2.1 Deposition . . . 2
  - 1.2.2 Drive-In . . . 2
  - 1.2.3 Oxidation . . . 2
  - 1.2.4 Diffusion Profile . . . 2
  - 1.2.5 Lifetime and Implied I-V curve . . . 3
- 1.3 Objectives . . . 4
  - 1.3.1 Problem Statement 1 . . . 4
  - 1.3.2 Problem Statement 2 . . . 4

2 Data Creation 5
- 2.1 Pipelined Web Services . . . 5
  - 2.1.1 Standard Mapper . . . 6
  - 2.1.2 Sentaurus Process . . . 6
  - 2.1.3 Conversion to *.JSON . . . 6
  - 2.1.4 Lifetime Simulation by PC1D . . . 6
  - 2.1.5 Post Processing of Lifetime . . . 6
- 2.2 Data Visualization . . . 6
  - 2.2.1 Problem Statement 1 . . . 7
  - 2.2.2 Problem Statement 2 . . . 8

3 Methodology: Building the Models 9
- 3.1 Problem Statement 1 . . . 9
  - 3.1.1 Scikit-learn and Random Forest . . . 9
  - 3.1.2 Model structure . . . 10
- 3.2 Problem Statement 2 . . . 11
  - 3.2.1 Neural Networks and TensorFLow (TF) . . . 11
  - 3.2.2 Architecture of model . . . 12

4 Results, Conclusion & Future work **15**
4.1 Results . . . 15
4.1.1 Problem Statement 1 . . . 15
4.1.2 Problem Statement 2 . . . 15
4.2 Conclusion . . . 16
4.2.1 Problem Statement 1 . . . 16
4.2.2 Problem Statement 2 . . . 16
4.3 Future Work . . . 16

A Python script to create randomized parameters **18**

B Python script for Step1 in Pipeline of Webservices **19**

C Python script for Steps3-7 in Pipeline of Webservices **21**

D Problem Statement 1 **24**

E Python Script for Dataset creation in Problem Statement2 **26**

F Problem Statement 2 **28**

G Bibliography **32**

H Slides used for Presentation **33**

Chapter 1

Introduction

1.1 Need for optimization in Solar Energy Research

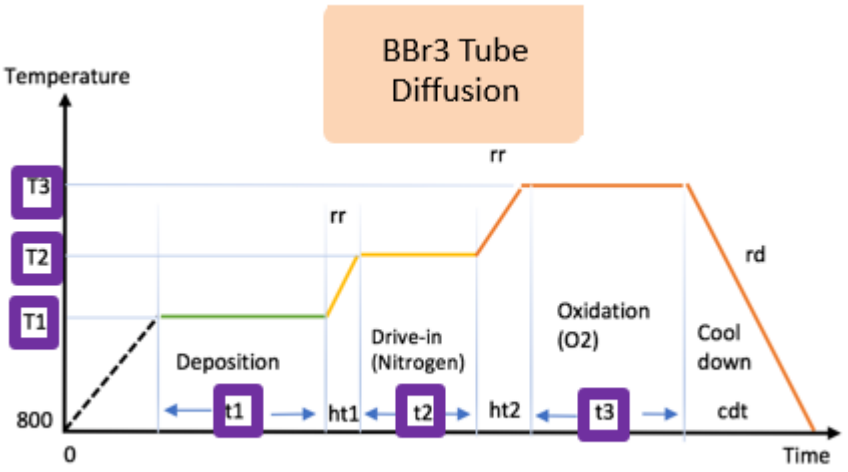
In the solar industry, there is a struggle to climb up the slope of efficiency in solar modules. Clean rooms strive to alter the parameters in the processes of manufacture to get better solar modules, with a larger portion of energy from sunlight converted to electricity via photovoltaics. We focus on optimization of one process, Boron Tube Diffusion, used for manufacture of a solar cell precursor.

1.2 Boron Tube Diffusion Process

A generalized 3-Step Boron Tube Diffusion Process has many parameters which have an effect on the final implied efficiency of the solar cell. We focus on tuning six of these, three temperatures and three lengths of time.

The process consists of 3 Steps:
1. Deposition
2. Drive-In
3. Oxidation

A generalized 3-Step process is depicted below:



The variables T1, T2, T3, ti1, ti2, and ti3 are varied for diffusion profile optimization.

1.2.1 Deposition

Covering the silicon wafer with a Borosilicate glass (BSG) layer and ramping up the temperature to T1 (Deposition Temperature) and allowing the dopants to enter the silicon wafer, for a time duration ti1.

1.2.2 Drive-In

Purging the oxygen in the chamber, scraping of the BSG layer and ramping up the temperature to T2 (Drive-In Temperature) to drive the dopants further into the wafer. This temperature is maintained for time duration ti2.

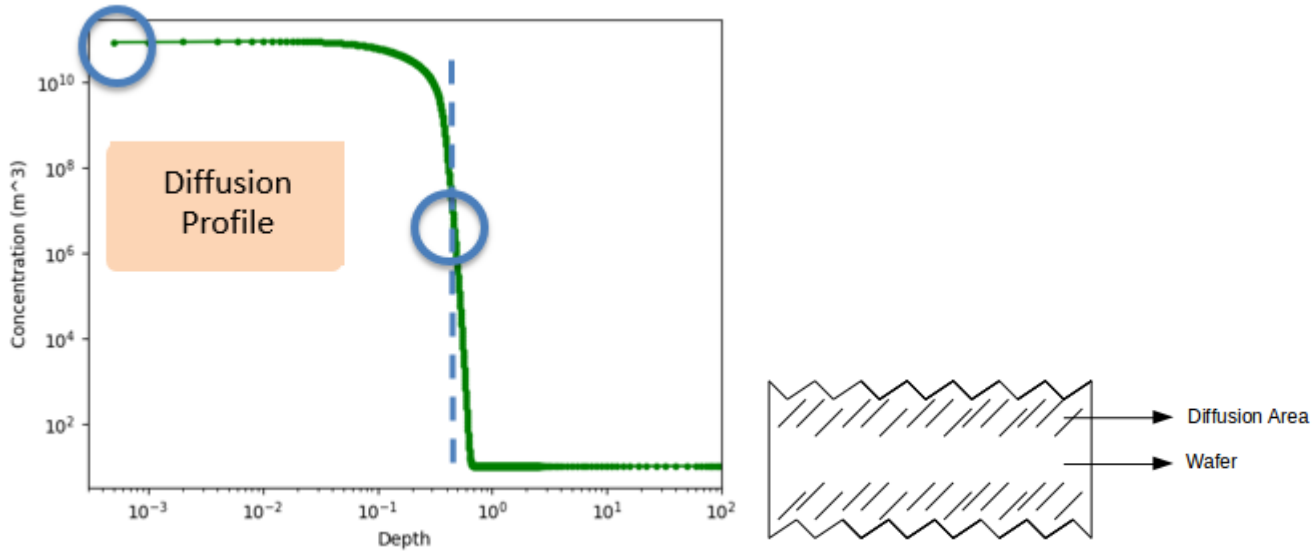
1.2.3 Oxidation

Releasing oxygen into the chamber, and further ramping up the temperature to T3 (Oxidation Temperature) for a time duration ti3, followed by cooling.

All other properties, including wafer properties are kept constant.

1.2.4 Diffusion Profile

Once the wafer has cooled down, we may measure a Diffusion Profile Graph, which plots the concentration at each depth within the wafer.

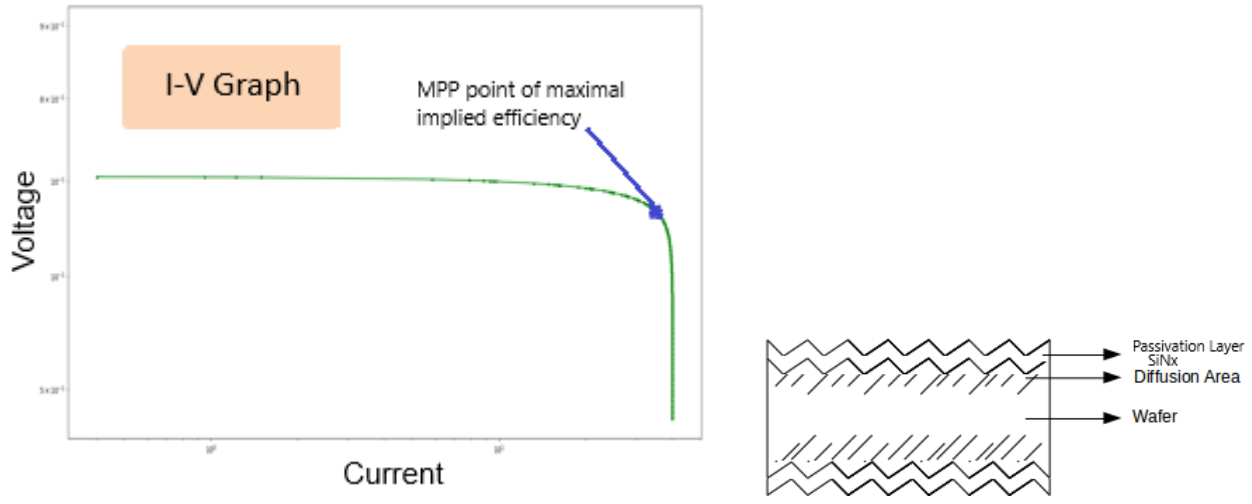


The circles represent values of dopant concentration at the surface of the silicon wafer, and the depth reached by 90% of the dopants.

Using this graph we can also calculate the sheet resistance indicating the ability of the diffused wafer in carrying the electric current.

1.2.5 Lifetime and Implied I-V curve

After a subsequent passivation layer deposition(using SiN_x) we can measure the lifetime of the sample, and further convert this data to obtain the implied efficiency of the diffused and passivated wafer it will make. Implied efficiency is simply the greatest product of I and V on the implied I-V curve. It indicates the maximum efficiency one could possibly obtain.

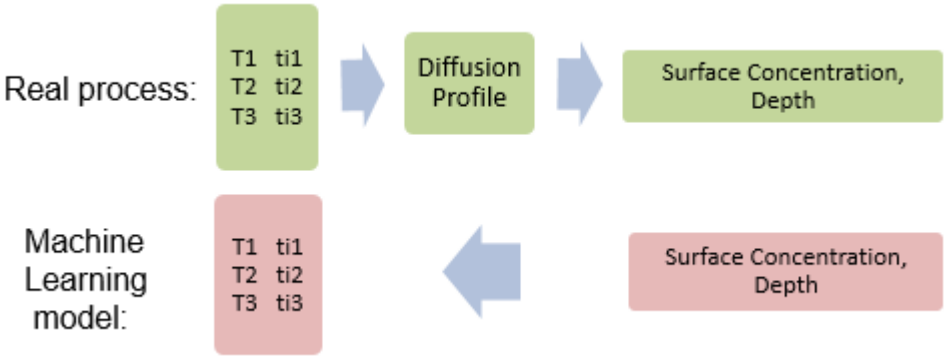


1.3 Objectives

1.3.1 Problem Statement 1

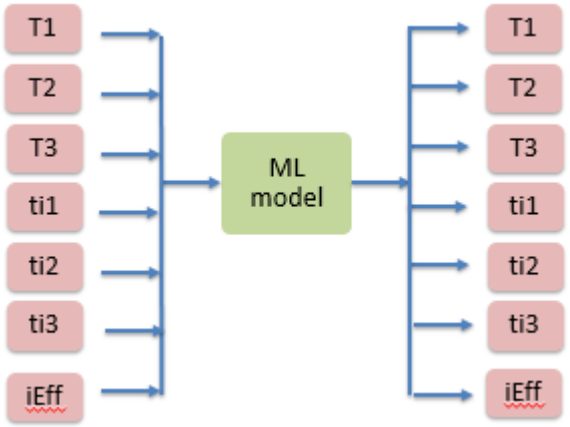
Our first objective is to predict the six process parameters, given two characteristics of the Diffusion Profile:

1. The dopant concentration at the surface
2. Depth reached by 90% of dopant concentration



1.3.2 Problem Statement 2

Given a set of 6 process parameters along with their implied efficiency, our second objective is to predict a new set of six parameters, along with the value of higher implied efficiency they'd achieve.



Chapter 2

Data Creation

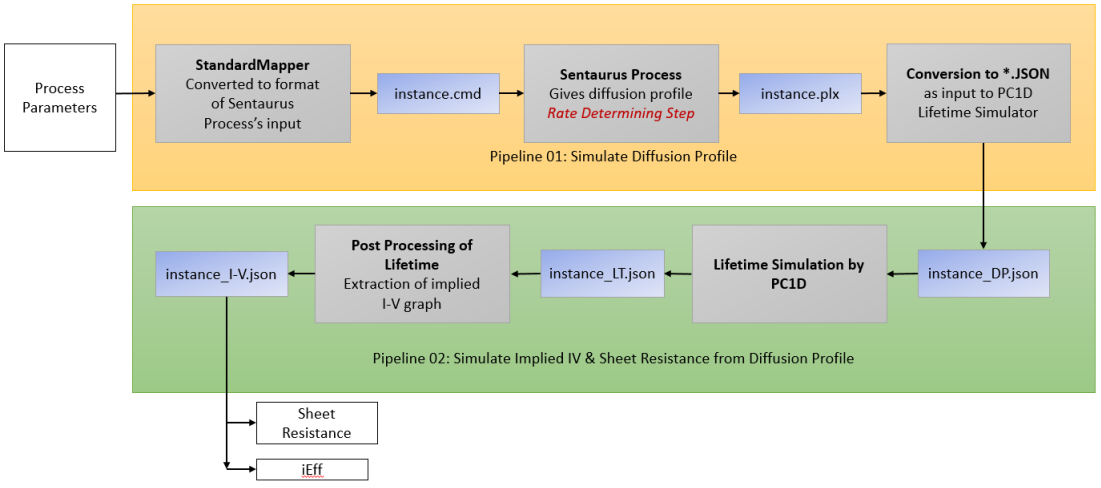
In order to build a machine learning model for prediction, we require true measurement data, which is tedious to gather. There are several simulation programs available which mimic real processes, and can be used for simulating a large dataset for training the model. A pipeline of webservices was created to collect the implied efficiency values from a set of input parameters, in a series of steps.

At a later stage, this artificial data will be combined with true measurement data, with a higher weight given to the true measured data samples in training.

2.1 Pipelined Web Services

Artificial data was generated using a series of simulation web services, with the output of one service being the input of the next.

Given below is a flowchart depicting the data generation. (Appendix B, C)



2.1.1 Standard Mapper

The six parameters are fed into StandardMapper, which produces a "instance.cmd" file to be read by Sentaurus Process.

2.1.2 Sentaurus Process

The Sentaurus process mimics a generalized 3 step Boron Tube Diffusion Process, and gives the Diffusion Profile in the form of "instance.plx" file. This is the slowest step, taking 4-5 minutes per sample.

2.1.3 Conversion to *.JSON

The "instance.plx" file is converted to "instance_DP.JSON" file, to be read by PC1D program for lifetime simulation.

2.1.4 Lifetime Simulation by PC1D

PC1D reads the diffusion profile data and calculates lifetime curve, output is a "instance_LT.JSON" file.

2.1.5 Post Processing of Lifetime

Lifetime data is read by Post Processing Program, which produces a graph of implied current v/s implied voltage, in the form of "instance_I-V.JSON".

The implied efficiency of the sample is calculated from the implied I-V curve. Sheet Resistance is calculated from the Diffusion Profile.

2.2 Data Visualization

The two problem statements require different subsets of the data generated from the pipeline. For the purpose of creating artificial datapoints, we require the 6 process parameters at the start of the pipeline. Once this data has been created, a subset of its attributes are used for the inputs and outputs of the machine learning models in the two problem statements.

2.2.1 Problem Statement 1

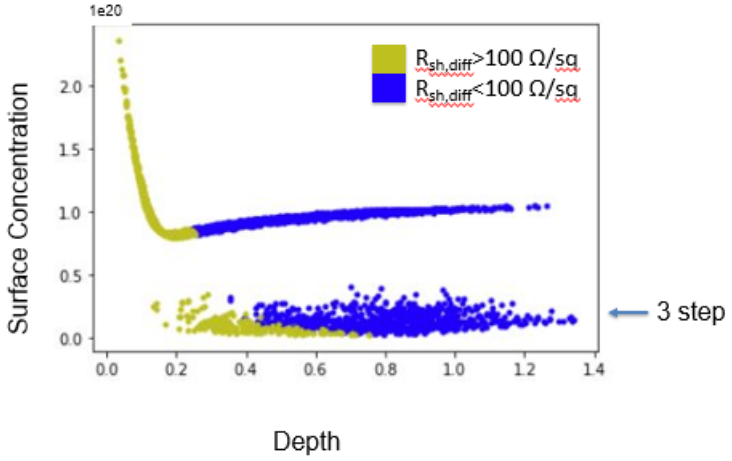
For problem statement 1, there was a constraint that the sheet resistance of the sample should be less than 100 Ω/sq, in order to make the sample "contactable" i.e. the metal probes which are to collect the current from the solar cell surface, will be able to carry out their job. If this value exceeds 100 Ω/sq, making a good contact to the sheet becomes difficult by screen printed Ag(silver).

Temperatures T1, T2 and T3 were varied in step sizes of approximately 40 ° across the permissible values between 800 ° Celsius and 1050 ° Celsius. Time durations ti1, ti2 and ti3 were varied in step sizes of 10 minutes between 10 and 60 minutes.

All possible combinations of the above valued process parameters were taken as input to the pipeline, resulting in a total of 6255 points. It took 2 weeks, with the following attribute ranges:

Input Data	min	max	range
90%Depth (µm)	0.255	1.45	1.19
SurfConc (cm^(-3))	2.14E+18	2.36E+20	2.34E+20

Additional Data	min	max	range
iEff (%)	23.6	25.75	2.15
SheetRes (Ohm/sq)	14.46	99.94	85.47



There was clustering observed in the data, which could be shown to be due to the types of Boron Tube Diffusion Processes. As described earlier, a 3 Step Process has 3 temperatures and 3 time durations of significance. In a 2 Step and 1 Step process, there are 2 plateaus and 1 plateau in the Time v/s Temperature plot of the process. i.e. The temperatures are ramped up twice and once respectively. These types of processes are barely used in industry. The 3-Step Process is most commonly used.

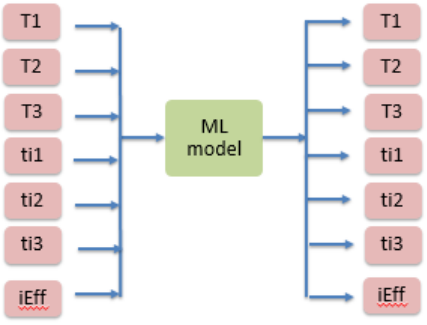
The cluster at the bottom of the graph mapping depth v/s surface concentration contains the points which belong to a 3-Step Process.

2.2.2 Problem Statement 2

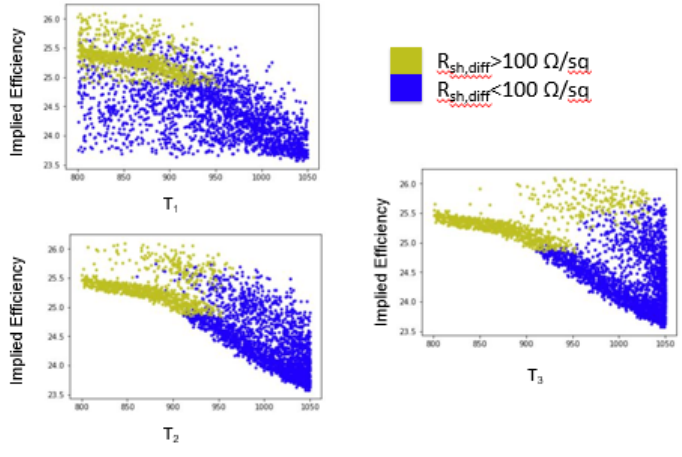
3000 points were freshly created over a period of 3 weeks and added to the dataset created for the first problem statement. Here, the values of 6 process parameters were completely randomized with uniform probability over their permissible ranges of 800 to 1050 ° Celsius for temperature and 10 to 60 minutes for time. (python script for randomized parameter generation can be found in Appendix A.)

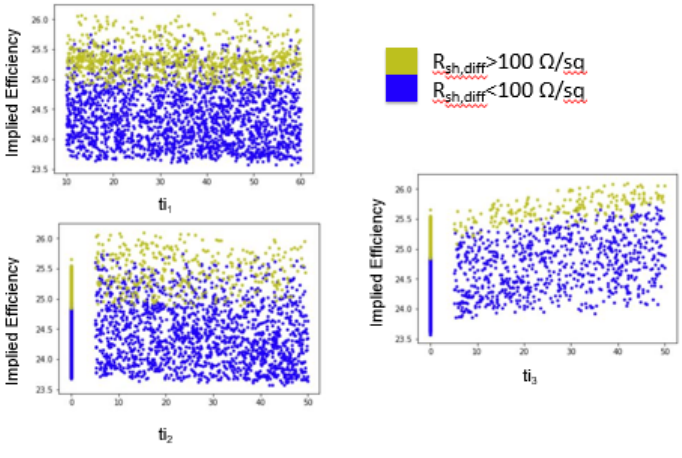
Here, the input and output shapes are the same for the machine learning model, consisting of the 6 process parameters and the value of implied efficiency. Input sample's sheet resistance may exceed 100 Ω/sq, but the output sample again, must be contactable.

The resulting attributes spanned the following ranges:



Inputs and Outputs	min	max
T1 (°C)	800	1050
T2 (°C)	800	1050
T3 (°C)	800	1050
ti1 (mins)	10	60
ti2 (mins)	0	50
ti3 (mins)	0	50
iEff (%)	23.56	26.17
SheetRes (Ohm/sq)	14.46	403.03





Chapter 3

Methodology: Building the Models

3.1 Problem Statement 1

3.1.1 Scikit-learn and Random Forest

Scikit-learn is a python library, widely used for implementation of machine learning models. It supports a fanfare of different algorithms for regression as well as classification, including Support Vector Machines, Multilayer Perceptron and Random Forest to name some of the powerful ones.

Regression done by a Random Forest (RF) is used when the prediction cannot lie outside of the range covered in the training dataset. It is a good way to minimize skewed prediction due to a few stray data points, and it also easily evades the problem of overfitting. This model was chosen for all six process parameter prediction, with varied number of trees in the forest, and varied maximum depth of each tree.

The RF model is a collection of decision trees, with the mean of predictions from all trees being the final prediction. Each tree is trained on a random subset of the complete dataset, and works by separating the data into smaller sized collections, not necessarily of the same size but of the lowest variance within a collection. The boundary for bifurcations is the value for an attribute of the dataset which divides the data into two subsets of lowest variance. The attribute is randomly chosen at each node.

As explained, there is a high degree of randomness in the algorithm, including the training subset of each tree, and the attribute chosen at each node. This makes the model robust against overfitting.

The technique of minimizing variance at each node makes the model extremely accurate. A new prediction is made by allowing each tree to classify the given test data point to one of its leaves. The mean of the training data points which reached that leaf is taken to be that tree's prediction.

3.1.2 Model structure

Six RF regression models were built serially, one for each process parameter, with an increasing featureset by including the parameters already predicted before it.

The order of prediction was built by checking individual accuracy using only surface concentration and depth as features. The order of individual accuracy was found to be:

T3 99.65% >
ti3 99.06% >
T2 98.74% >
T1 96.99% >
ti2 96.73% >
ti1 87.09%

Number of trees in the forest ranged from 1000 to 5000.
Maximum depth of trees ranged from 20 to 10

Accuracy of prediction was measured by using the predicted 6 parameters as input in the data generation pipeline, and the value of surface concentration and depth thus obtained were compared with the values prior to prediction.

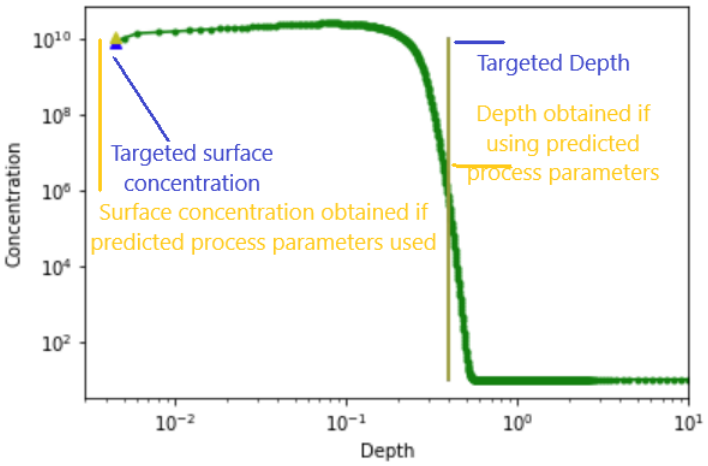
Accuracy was ascertained at:
Surface Concentration: 86.23%
Depth at 90% Concentration: 94.61%

(Code for Problem Statement 1: Appendix D)

Screenshot of one example prediction is shown below:

```
pred(1.05e10,.4)
Predicting ti3...
Done.
Predicting T3...
Done.
Predicting T2...
Done.
Predicting T1...
Done.
Predicting ti2...
Done.
Predicting ti1...
Done.

Predicted parameters are:
T1:   850.761
T2:   958.6925
T3:   966.165
ti1:  25.277
ti2:  26.09
ti3:  18.755
```




```
Accuracy of this prediction:

Error in Surface Concentration predcition =  2434042000.0
Percentage error =  0.23181352381
Error in Depth predcition =  0.0011075
Percentage error =  0.276875
```

Predicting the 6 process parameters which shall lead to a surface concentration of $1.05 \text{x} 10^{19}$ cm^{-3} and depth of 0.4 μm.

3.2 Problem Statement 2

Given a set of 6 parameters and the implied efficiency they achieve, a new set of parameters was predicted, with the promise of achieving a better implied efficiency.

3.2.1 Neural Networks and TensorFLow (TF)

Neural Networks are the front-runners in machine learning problem statements today, with immense power and even untapped potential within their architectures. We explore the use of a 3 layer feed-forward neural network implemented in TensorFlow, with 7, 10 and 7 nodes respectively.

Tensorflow is the most used platform for building a neural network, with a global community of experts and ready ways for serving up a trained model. This was perfect for our problem statement, for easy, recurrent use of the model, once trained.

Tensorflow allows complex matrix multiplications that are computationally expensive by using low-level optimization wrapped in python. These operations are the crux of training a neural

network, repeatedly done for more than 4 orders of magnitude. Other platforms do not provide support for building and tweaking neural networks to the extent that TF does.

3.2.2 Architecture of model

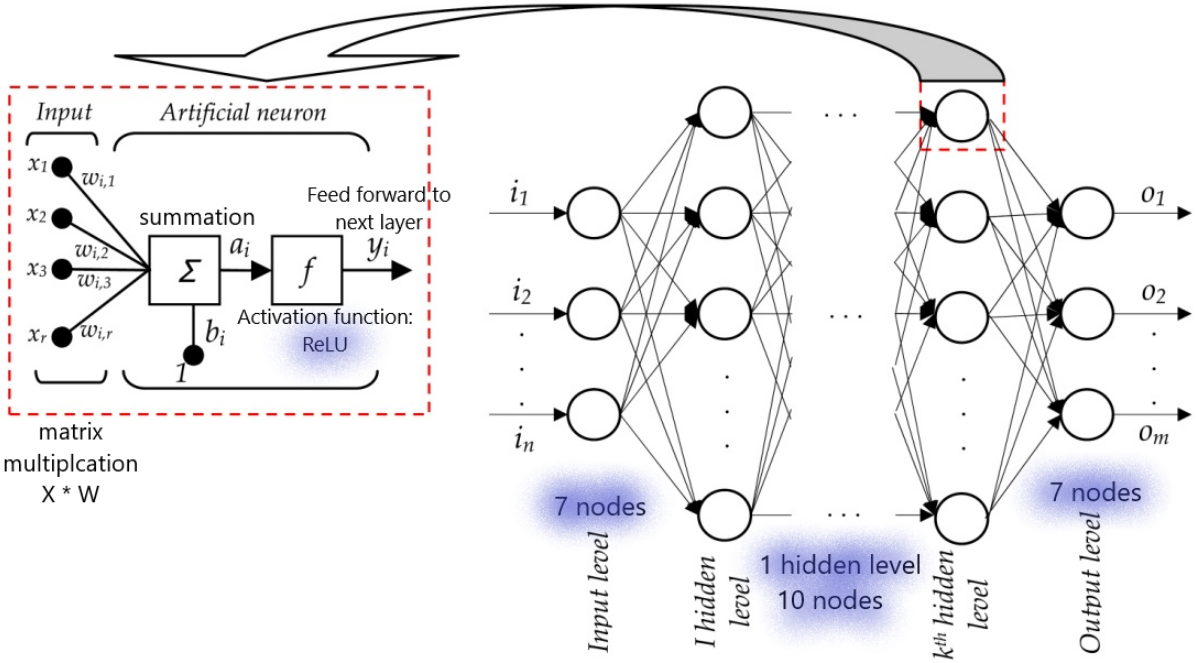
The input layer consisted of the input set of parameters and their implied efficiency value. The output layer had a similar structure, and the resulting implied efficiency was to be higher than the one at input. The hidden layer consisted of 10 nodes. The activation function used was Rectified Linear Unit (RELU) for the hidden layer.

The shape of the dataset looked like:
X: T1, T2, T3, ti1, ti2, ti3, iEff
Y: T1', T2', T3', ti1', ti2', ti3', iEff'
Where iEff' $>$ iEff

It was built by arranging all 9727 points in ascending order of implied efficiency, and pairing the i^{th} sample with the i+1^{th}, i+2^{th}, i+3^{th} and i+4^{th} sample to create an X-Y pair. (AppendixE) (A visual representation of this process can be found on the Appendix H slides page 13.)

This generated a dataset of 36227 instances, of which training, testing, and validation sets consisted of 20287, 8695 and 7245 instances respectively. (ratio being 80:20 for training+testing to validation, and 70:30 for training to testing)

Given below is a visual representation of the architecture used, with choices for number of nodes, number of layers and activation function highlighted in blue.



Loss was mean squared error between prediction and true Y value. Optimization was done by updating weights in proportion to gradient of loss with respect to weight at that point by a factor of the learning rate.

The model was trained for 2000 epochs, and achieved the objective of prediction of better parameters in the direction of increasing implied efficiency, across the validation set.

(Code for Problem Statement 2: Appendix F)

A screenshot of an input to the model (from the unseen validation set, already preprocessed) and the predicted parameters is shown below. The order of output values is:
T1, T2, T3, ti1, ti2, ti3, iEff

Predicting 6 new process parameters, given the 6 process parameters:
T1 (Deposition temperature) = 840.088 ° C
T2 (Drive-In temperature) = 855.963 ° C
T3 (Oxidation temperature) = 855.963 ° C
ti1 (Deposition time) = 24 mins
ti2 (Drive-In time) = 14 mins
ti3 (Oxidation time) = 0 mins
and the corresponding implied efficiency they achieve = 25.12%

The prediction of 6 new process parameters consist of:
T1 (Deposition temperature) = 866.891 ° C
T2 (Drive-In temperature) = 935.506 ° C
T3 (Oxidation temperature) = 1027.847 ° C
ti1 (Deposition time) = 28.739 mins
ti2 (Drive-In time) = 29.44 mins
ti3 (Oxidation time) = 29.173 mins
and the corresponding implied efficiency they achieve = 25.141%

Checking the value of implied efficiency by entering these process parameters in the pipeline, we find the efficiency to be 25.799%, showing an increase of 0.658%

In [94]:
```
a = test_set[0:7,9].reshape(7,1)
# a = np.array([800., 900., 1000., 5., 5., 25.,25.]).reshape(1,7)
# a -= np.mean(X_, axis = 0)
# a /= np.std(X_, axis = 0)
print(a)
```

```
[[-0.7375146 ]
 [-1.58601908]
 [-2.33719838]
 [-0.59005601]
 [-0.81421988]
 [-1.08377492]
 [ 0.53685793]]
```

In [95]:
```
mean = [  890.68657398,   957.42422753,   995.9211774 ,    32.063477  ,
          27.15144739,    18.15487214,    24.92921596,   891.43500688,
         965.83692678,  1023.16627526,    31.31001959,    28.81302863,
          23.21309011,    24.92604088]
var = [ 68.60648957,  63.97213147,  59.88279466,  13.66745704,  16.27517777,
  16.75151529,   0.55665669,  68.89530017,  59.48288375,  31.67465839,
  13.40834459,  14.62242957,  17.1366787,    0.54971325,]
with tf.Session() as sess1:
#     sess.run(tf.global_variables_initializer())
    saver.restore(sess1, dst)
    pr_dict = {x_go: a}
    prediction = sess1.run([y_go], feed_dict = pr_dict)
    pred = prediction[0].T * var[7:]
    prd = pred + mean[7:]
    print("Prediction is: ")
    print(prd[0])
```

```
INFO:tensorflow:Restoring parameters from ../models_dec/model2.ckpt
Prediction is:
[  866.89132394   935.50569827  1027.84664657    28.73902519    29.43985057
    29.17264197    25.1411198 ]
```

Chapter 4

Results, Conclusion & Future work

4.1 Results

4.1.1 Problem Statement 1

A single prediction (given inputs of surface concentration and depth) consisted of calling the regressor model for ti3, and storing that value. Then calling the regressor model for T3 with inputs of surface concentration, depth and ti3. The next call was to regressor for T2 with inputs of surface concentration, depth, ti3 and T3. So on, till all 6 parameters had been predicted.

Accuracy of prediction was calculated by using these predicted values of process parameters and generating the surface concentration and depth from the dataset generation pipeline. This was repeated for 2000 samples, and the coefficient of determination was taken to be the accuracy of prediction.

Problem statement 1 achieved 86.23% accuracy in backwards prediction of Surface Concentration. Problem statement 1 achieved 94.61% accuracy in backwards prediction of Depth at 90% Concentration.

4.1.2 Problem Statement 2

Accuracy is differently defined for Problem statement 2, where the model successfully predicted a set of parameters which gave a higher implied efficiency, across all instances in the validation dataset (7245 instances). This result was validated by observing proximity to the measured value of implied efficiency, when predicted process parameters were fed back into the generation pipeline.

4.2 Conclusion

4.2.1 Problem Statement 1

An accurate model for predicting process parameters is useful for researchers in the Solar Energy domain, in order to correctly calibrate the process to obtain a certain diffusion parameters, characterized by its surface concentration and depth.

The trained model is ready for serving to research institutes via a web platform called "X-Solar Hetero."

Furthermore, this technique for prediction has only been tested on one process, i.e. Boron Tube Diffusion. It may be applied to many other processes, and may be a good tool for the research community.

4.2.2 Problem Statement 2

This is a novel technique used for optimization, given the value to be optimized, and features, without prior knowledge of their mathematical relationship. The neural network architecture is good at learning the relationship between input and output, which has tremendous potential for use in optimization strategies.

It doesn't however, allow for analysis of the relationship itself.

This approach should bear consequence in many applications, should it be proven successful in climbing to unseen values of efficiency, for the problem statement at hand.

4.3 Future Work

1. Integration of the models in problem statement 1 as web services on X-Solar Hetero.

2. Use of secondary regressor models instead of pipeline to determine accuracy of prediction, which are extremely accurate.

3. Future work includes chopping off the top 20% of data in terms of best implied efficiency, and iterating over the samples below that as input to model 2, to see if there is a climb up to the unseen true global maximum of implied efficiency.

4. Future work also includes retraining model 2 on real measurement data, and building a system for retraining the model as and when real data is more readily available. Two concepts in deep learning are applicable here:
i) Transfer Learning, wherein only the latter layers of the neural network are replaced and retrained with real measured data.
ii) Recurrent Neural Networks, which contain Long Short Term Memory units, capable of storing information from instances of the past.

Appendix A

Python script to create randomized parameters

```
import numpy as np
count = 3000
batch = 'rand'
np.random.seed(0)
maximum = {
    'T1': 1050, 'ti1': 60,
    'T2': 1050, 'ti2': 50,
    'T3': 1050, 'ti3': 50
}
minimum = {
    'T1': 800, 'ti1': 10,
    'T2': 800, 'ti2': 5,
    'T3': 800, 'ti3': 5
}
num_of_steps = np.random.randint(low = 1, high = 4, size = count)

T1 = np.random.rand(count) * (maximum['T1'] - minimum['T1']) + minimum['T1']

ti1 = np.random.rand(count) * (maximum['ti1'] - minimum['ti1']) + minimum['ti1']
ti2 = np.random.rand(count) * (maximum['ti2'] - minimum['ti2']) + minimum['ti2']
ti3 = np.random.rand(count) * (maximum['ti3'] - minimum['ti3']) + minimum['ti3']

T2 = np.random.rand(count)
T3 = np.random.rand(count)

for i in range(0,count):
    T2[i] = T2[i] * (maximum['T2'] - max(minimum['T2'],T1[i])) + max(minimum['T2'],T1[i])
    T3[i] = T3[i] * (maximum['T3'] - max(minimum['T3'],T2[i])) + max(minimum['T3'],T2[i])
with open('Z:/rand.csv', "a") as f:              ## Name of Dataset
    f.write('Batchname,InstanceNo,num_of_steps,T1,T2,T3,ti1,ti2,ti3\n')
    for i in range(0,count):
f.write(batch + ',' + str(i) + "," + str(num_of_steps[i]) + "," + str(T1[i]) + \
    "," + str(T2[i]) + "," + str(T3[i]) + "," + str(ti1[i]) + "," + \
    str(ti2[i]) + "," + str(ti3[i]) + "\n")
```

Appendix B

Python script for Step1 in Pipeline of Webservices

```
import pandas as pd
import requests
import os
import sys

d = pd.read_csv('Z:/rand.csv')
batchname = "rand"
err_count = 0
count = 3000
re_batchsize = 500
re_batchnum = 0
os.makedirs('Z:/' + batchname)
for i in range(1,int(count/re_batchsize) + 2,1):
    os.makedirs('Z:/' + batchname + "/" + str(i))
for i in range(0,len(d['InstanceNo'])):
    re_batchnum = i/re_batchsize + 1
    num_step = d["num_of_steps"][i]
    T1 = d["T1"][i]
    T2 = d["T2"][i]
    T3 = d["T3"][i]
    ti1 = d["ti1"][i]
    ti2 = d["ti2"][i]
    ti3 = d["ti3"][i]
    Params_payload = {
        'operation': 'StandardmapperNUSServer',
        'FileInput': '',
        'Testing': 'false',
        'UserID': batchname + str(d["InstanceNo"][i]),
        'NoOfSteps': 3,
        'isTextured': 'true',
        'DepositionTempT1': str(T1),
        'DepTimet1': str(ti1),
        'DriveinTempT2': str(T2),
        'DriveinTimet2': str(ti2),
        'OxidationTempT3': str(T3),
        'OxidationTimet3': str(ti3),
        'isO2purgedDuringCoolD': 'false',
```

```
        'OutFileName': batchname + str(d["InstanceNo"][i]),
        'ExecutionTime': '',
        'ArchitectureByID': '',
        'AccessID': '',
        'URI': '',
        'isZip': '',
        'inZipRefFile': '',
        'SentaurusPassword': '',
        'SentaurusUsername': '',
        'format': 'OutputForm'
    }
    url = 'http://127.0.0.1:8080//SentaurusProcess//API.m?operation=StandardmapperNUSServer'
    if num_step <= 2:
        Params_payload['operation'] = 'StandardMapper2'
        Params_payload['NoOfSteps'] = 2
        del Params_payload['OxidationTimet3']
        del Params_payload['OxidationTempT3']
        url = 'http://137.132.123.119/SentaurusProcess/API.m?operation=StandardMapper2'
    if num_step == 1:
        Params_payload['operation'] = 'StandardMapper1'
        Params_payload['NoOfSteps'] = 1
        del Params_payload['DriveinTempT2']
        del Params_payload['DriveinTimet2']
        url = 'http://137.132.123.119/SentaurusProcess/API.m?operation=StandardMapper1'
    try:
        r = requests.get(url, auth=(user, password),params=Params_payload,stream=False, timeout=120)
        a = r.raise_for_status()
        #print(r.url)
    except:
        print('Creating .cmd files: Error in file ' + batchname + '/' + \
            str(re_batchnum) + '/' + batchname + str(d["InstanceNo"][i]))
        e = sys.exc_info()[0]
        print(str(e))
        err_count += 1
        if err_count==5:
            print('Five errors! ABORTING')
            sys.exit(0)
        continue
    with open('Z:/' + batchname + '/' + str(re_batchnum) + '/' + batchname + \
        str(d["InstanceNo"][i]) +".cmd", "a") as f:
            f.write(r.text)
```

Appendix C

Python script for Steps3-7 in Pipeline of Webservices

```
import requests
import pandas as pd
import time
import sys
import numpy as np
count = 2
#d = pd.read_csv('Z:\Rolf_test.csv')

def getOutput(parameters,uri,stepName,batch,no):
    global err_count
    try:
        r = requests.get(uri, auth=(user, password),params=parameters,stream=False, timeout=120)
        a = r.raise_for_status()
       # print(r.url)
    except:
        print(stepName + ': Error in file number ' + batch + no)
        e = sys.exc_info()[0]
        print(str(e))
        err_count += 1
        if err_count==5:
            print('Five errors! ABORTING')
            sys.exit(0)
        return False
    with open('D:/filehost/fileapp/public/' + stepName + '/' + batch + '/' + batch + no +".json", "a")
        f.write(r.text)
   # time.sleep(2)
    return True

err_count = 0

for i in range(0,count):#len(d['InstanceNo'])):
    batchname = 'nov'#d['Batchname'][i]
    num = str(i)#str(d['InstanceNo'][i])

### PLX to DiffProfile
    step = 'DiffProfile'
    url = 'http://127.0.0.1:8080//SentaurusProcess//API.m?operation=\
```

```
        PlxToStandardMeasurementDiffusionProfile'
    Params_payload = {
    'operation': 'PlxToStandardMeasurementDiffusionProfile',
    'FileInput': '"D:\\\\filehost\\\\fileapp\\\\public\\\\PLXfiles\\\\' + \
        batchname + '\\\\' + batchname + num + '.plx"',
    'UserID': '',
    'ExecutionTime': '',
    'ArchitectureByID': '',
    'Testing': 'false',
    'AccessID': '',
    'URI': '',
    'isZip': '',
    'inZipRefFile': '',
    'DiffusionType': 'p',
    'format': 'OutputForm'
    }
    prev_done = getOutput(Params_payload,url,step,batchname,num)

### Sheet Resistance from DiffProfile
    step = 'SheetRes'
    url = 'http://127.0.0.1:8080//SheetResistanceCalculator//API.m?operation=SheetResistance'
    Params_payload = {
    'operation': 'SheetResistance',
    'FileInput': '"D:\\\\filehost\\\\fileapp\\\\public\\\\DiffProfile\\\\' + \
        batchname + '\\\\' + batchname + num + '.json"',
    'UserID': '',
    'ExecutionTime': '',
    'ArchitectureByID': '',
    'Testing': '',
    'AccessID': '',
    'URI': '',
    'isZip': '',
    'inZipRefFile': '',
    'format': 'OutputForm'
    }
    if prev_done:
        prev_done = getOutput(Params_payload,url,step,batchname,num)
    else:
        continue

### SurfMax90Conc from DiffProfile
    step = 'SurfMax90Conc'
    url = 'http://127.0.0.1:8080//PostProcessingDiffusionProfile//API.m?operation=ProcessedJSON'
    Params_payload = {
    'operation': 'ProcessedJSON',
    'FileInput': '"D:\\\\filehost\\\\fileapp\\\\public\\\\DiffProfile\\\\' + \
        batchname + '\\\\' + batchname + num + '.json"',
    'UserID': '',
    'ExecutionTime': '',
    'ArchitectureByID': '',
    'Testing': '',
    'AccessID': '',
    'URI': '',
    'isZip': '',
    'inZipRefFile': '',
    'format': 'OutputForm'
    }
```

```
    if prev_done:
        prev_done = getOutput(Params_payload,url,step,batchname,num)
    else:
        continue

### DiffProfile to StdLifetime
    step = 'StdLifeTime'
    url = 'http://127.0.0.1:8080//PC1Dv6Online//API.m?operation=StandardLifetime'
    Params_payload = {
    'operation': 'StandardLifetime',
    'FileInput': '"D:\\\\filehost\\\\fileapp\\\\public\\\\DiffProfile\\\\' + \
        batchname + '\\\\' + batchname + num + '.json"',
    'UserID': '',
    'ExecutionTime': '',
    'ArchitectureByID': '',
    'Testing': 'false',
    'AccessID': '',
    'URI': '',
    'isZip': '',
    'inZipRefFile': '',
    'FrontSn': '1.2484',
    'FrontSp': '1.2484',
    'RearSn': '1.2484',
    'RearSp': '1.2484',
    'Thicknessw': '0.018',
    'BulkDopTyp': '0',
    'BulkDop': '1e+015',
    'BulkRecombn': '0.005',
    'format': 'OutputForm'
    }
    if prev_done:
        prev_done = getOutput(Params_payload,url,step,batchname,num)
    else:
        continue

### StdLifetime to ImpIV
    step = 'impIV'
    url = 'http://127.0.0.1:8080//PostProcessingLifetime//API.m?operation=ImpliedCurntVltg'
    Params_payload = {
    'operation': 'ImpliedCurntVltg',
    'scale': '"lin"',
    'depvar2': '',
    'GenerationCurrent': '40',
    'FileInput': '"D:\\\\filehost\\\\fileapp\\\\public\\\\StdLifeTime\\\\' + \
        batchname + '\\\\' + batchname + num + '.json"',
    'UserID': '',
    'ExecutionTime': '',
    'ArchitectureByID': '',
    'Testing': '',
    'AccessID': '',
    'URI': '',
    'isZip': '',
    'inZipRefFile': '',
    'format': 'OutputForm'
    }
    if prev_done:
prev_done = getOutput(Params_payload,url,step,batchname,num)
```

Appendix D

Problem Statement 1

```
from sklearn.ensemble import RandomForestRegressor
import numpy as np
import pandas as pd

d = pd.DataFrame.from_csv('useful.csv')
x = d[['surfConc', '90%Depth']].copy()
X = np.array(x)

y_t3 = d['ti3'].copy()
y23 = np.array(y_t3)
regr6 = RandomForestRegressor(max_depth=26, n_estimators = 1000, random_state=0)
regr6.fit(X,y23)
#print(regr.feature_importances_)
regr6.score(X,y23)

x = d[['surfConc', '90%Depth', 'ti3']].copy()
X = np.array(x)

y_T3 = d['T3'].copy()
y13 = np.array(y_T3)
regr3 = RandomForestRegressor(max_depth=20, n_estimators = 1000, random_state=0)
regr3.fit(X,y13)
#print(regr.feature_importances_)
regr3.score(X,y13)

x = d[['surfConc', '90%Depth', 'ti3', 'T3']].copy()
X = np.array(x)

y_T2 = d['T2'].copy()
y12 = np.array(y_T2)
regr2 = RandomForestRegressor(max_depth=100, n_estimators = 2000, random_state=0)
regr2.fit(X,y12)
#print(regr.feature_importances_)
regr2.score(X,y12)

x = d[['surfConc', '90%Depth', 'ti3', 'T3', 'T2']].copy()
X = np.array(x)

y_T1 = d['T1'].copy()
y11 = np.array(y_T1)
```

```
regr1 = RandomForestRegressor(max_depth=50, n_estimators = 5000, random_state=0)
regr1.fit(X,y11)
#print(regr.feature_importances_)
regr1.score(X,y11)

x = d[['surfConc', '90%Depth', 'ti3', 'T3', 'T2', 'T1']].copy()
X = np.array(x)

y_t2 = d['ti2'].copy()
y22 = np.array(y_t2)
regr5 = RandomForestRegressor(max_depth=31, n_estimators = 5000, random_state=0)
regr5.fit(X,y22)
#print(regr.feature_importances_)
regr5.score(X,y22)

x = d[['surfConc', '90%Depth', 'ti3', 'T3', 'T2', 'T1', 'ti2']].copy()
X = np.array(x)

y_t1 = d['ti1'].copy()
y21 = np.array(y_t1)
regr4 = RandomForestRegressor(max_depth=46, n_estimators = 5000, random_state=0)
regr4.fit(X,y21)
#print(regr.feature_importances_)
regr4.score(X,y21)


def pred(sc_act,dep_act): # sc in 1e+09
    print "Un momento, back predicting Process Parameters...\n\nPredicting ti3..."
    yolo = np.array([sc_act,dep_act]).reshape((1,-1))
    ti3 = float(regr6.predict(yolo))
    print "Done.\n\nPredicting T3..."
    yolo = np.insert(yolo,2,ti3, axis = 1)
    T3 = float(regr3.predict(yolo))
    print "Done.\n\nPredicting T2..."
    yolo = np.insert(yolo,3,T3, axis = 1)
    T2 = float(regr2.predict(yolo))
    print "Done.\n\nPredicting T1..."
    yolo = np.insert(yolo,4,T2, axis = 1)
    T1 = float(regr1.predict(yolo))
    print "Done.\n\nPredicting ti2..."
    yolo = np.insert(yolo,5,T1, axis = 1)
    ti2 = float(regr5.predict(yolo))
    print "Done.\n\nPredicting ti1..."
    yolo = np.insert(yolo,6,ti2, axis = 1)
    ti1 = float(regr4.predict(yolo))
    print "Done."
    yolo = np.insert(yolo,7,ti1, axis = 1)
    print "\n\nPredicted parameters are:\n\nT1:  ", yolo[0][5],"\nT2:  ", yolo[0][4]
    print "T3:  ", yolo[0][3], "\nti1: ", yolo[0][7], "\nti2: ", yolo[0][6]
    print "ti3: ", yolo[0][2]


pred(4e9,.65)
```

Appendix E

Python Script for Dataset creation in Problem Statement2

```
import numpy as np
# import tensorflow as tf
np.random.seed(0)
import random
import pandas as pd
# import matplotlib.pyplot as plt
#from sklearn import metrics
# import itertools

domain_ = pd.read_csv('novALL.csv')
xef = domain_['iEff']
domain_ = domain_.drop(['Batchname','InstanceNo','SheetRes','usable?','iVoC','surfConc',\
        'maxConc','90%Conc','90%Depth'], axis = 1)
x_ = np.array(domain_)
range_ = pd.read_csv('novUsable.csv')
yef = range_['iEff']
range_ = range_.drop(['Batchname','InstanceNo','SheetRes','iVoC','surfConc','maxConc',\
        '90%Conc','90%Depth'], axis = 1)
y_ = np.array(range_)


X = []
Y = []
j = len(yef) - 1
for i in range(len(xef)-1,-1,-1):
    if xef[i] >= yef[j]:
        X.append(x_[i])
        Y.append(y_[j])
        continue
    else:
        while xef[i] < yef[j]:
            j -= 1
        j+=1
    if j > (len(yef) - 4):
        for zz in range(j,len(yef)):
            X.append(x_[i])
            Y.append(y_[zz])
```

```
    else:
        for zz in range(j,j+4):
            X.append(x_[i])
            Y.append(y_[zz])

X_ = np.array(X)
Y_ = np.array(Y)
dataset = np.concatenate((X_,Y_,),axis =1)
print(dataset.shape)
dataset -= np.mean(dataset, axis =0)
dataset /= np.std(dataset,axis =0)
with open('dataset.csv', 'wb') as f:
    f.write(b'T1,T2,T3,ti1,ti2,ti3,iEff,newT1,newT2,newT3,newti1,newti2,newti3,newiEff\n')
    np.savetxt(f, dataset, delimiter=",")

datasetmv = np.concatenate((X_,Y_,),axis =1)
print np.mean(datasetmv, axis =0)
print np.std(datasetmv, axis =0)
```

Appendix F

Problem Statement 2

```
import numpy as np
import tensorflow as tf
np.random.seed(0)
import random
import pandas as pd
import matplotlib.pyplot as plt
#from sklearn import metrics
import itertools


all_df = pd.read_csv('dataset.csv') # preprocessing done beforehand
evaluate_df = all_df.sample(frac=0.2,random_state=200)
train_df = all_df.drop(evaluate_df.index)
test_df = train_df.sample(frac=0.3, random_state=200)
train_set = np.array(train_df.drop(test_df.index)).T
test_set = np.array(test_df).T
evaluate_set = np.array(evaluate_df).T
print evaluate_set.shape
print test_set.shape
print train_set.shape


N_tr, N_t, I, O, H = 20287, 8695, 7, 7, 10
w1 = tf.get_variable("w1", [H,I], initializer = tf.contrib.layers.xavier_\
        initializer(seed = 1))
b1 = tf.get_variable("b1", [H,1], initializer = tf.zeros_initializer())
w2 = tf.get_variable("w2", [O,H], initializer = tf.contrib.layers.xavier_\
        initializer(seed = 1))
b2 = tf.get_variable("b2", [O,1], initializer = tf.zeros_initializer())


x_tr = tf.placeholder(tf.float32, shape = (I, None))
y_tr = tf.placeholder(tf.float32, shape = (O, None))

h = tf.maximum(tf.add(tf.matmul(w1, x_tr),b1), 0)
A1 = tf.nn.leaky_relu(h)
y_pred = tf.add(tf.matmul(w2,A1),b2)

diff = y_tr - y_pred
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis = 1))
```

```
grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])


learning_rate = 1e-10
step = []
losses = []
acc = []
gap = []
i = 0


new_w1 = w1.assign(w1 - learning_rate * grad_w1)
new_w2 = w2.assign(w2 - learning_rate * grad_w2)
updates = tf.group(new_w1, new_w2)


x_t = tf.placeholder(tf.float32, shape = (I, None))
y_t = tf.placeholder(tf.float32, shape = (O, None))

h_t =  tf.maximum(tf.add(tf.matmul(w1, x_t),b1), 0)
A1_t = tf.nn.leaky_relu(h_t)
y_pred_t = tf.add(tf.matmul(w2, A1_t),b2)

diff_t = y_t - y_pred_t
loss_t = tf.reduce_mean(tf.reduce_sum(diff_t ** 2, axis = 1))


x_go = tf.placeholder(tf.float32, shape = (I,None))

h_go = tf.maximum(tf.add(tf.matmul(w1,x_go),b1), 0)
A1_go = tf.nn.leaky_relu(h_go)
y_go = tf.add(tf.matmul(w2,A1_go),b2)


saver = tf.train.Saver()
dst = "../models_dec/model2.ckpt"


with tf.Session() as sess1:
    tf.set_random_seed(1)               # to keep consistent results
#     seed = 3
    sess1.run(tf.global_variables_initializer())
#     saver.restore(sess1, dst)
    keys = [x_tr, y_tr]
    vals = [train_set[0:7], train_set[7:14]]
    lol = [x_t, y_t]
    vs = [test_set[0:7], test_set[7:14]]
    values = dict(zip(keys, vals))
    accs = dict(zip(lol, vs))
    for t in range(1):
#         print(i)
        if i%50 == 0:
            print(i)
        loss_val = sess1.run([loss, updates], feed_dict = values)
        step.append(i)
        losses.append(loss_val)
```

```
        i +=1
        accuracy = sess1.run([loss_t], feed_dict = accs)
#         print(loss_val)
#         print(accuracy)
        gap.append(loss_val[0] - accuracy[0])
        acc.append(accuracy)
#         acc.append(1-(accuracy/170.7666658987))
    save_path = saver.save(sess1, dst)
    print("Model saved in file: %s" % save_path)


# import time
with tf.Session() as sess:
#     sess.run(tf.global_variables_initializer())
    saver.restore(sess, dst)
    keys = [x_tr, y_tr]
    vals = [train_set[0:7], train_set[7:14]]
    lol = [x_t, y_t]
    vs = [test_set[0:7], test_set[7:14]]
    values = dict(zip(keys, vals))
    accs = dict(zip(lol, vs))
    for t in range(2000):
#         print(i)
        if i%500 == 0:
            print(i/500)
#             time.sleep(10)
        loss_val = sess.run([loss, updates], feed_dict = values)
        step.append(i)
        losses.append(loss_val)
        i +=1
        accuracy = sess.run([loss_t], feed_dict = accs)
#         print(loss_val)
#         print(accuracy)
        gap.append(loss_val[0] - accuracy[0])
        acc.append(accuracy)
#         acc.append(1-(accuracy/170.7666658987))
    save_path = saver.save(sess, dst)
    print("Model saved in file: %s" % save_path)



plt.plot(step, losses, 'g', step, acc, 'y')#, step, gap, 'b')
# plt.ylim(0,50)
plt.show()

plt.plot(step, gap, 'b')
# plt.ylim(-1,1)
plt.show()

a = test_set[0:7,9].reshape(7,1)
print(a)


mean = [  890.68657398,   957.42422753,   995.9211774 ,    32.063477  ,
           27.15144739,    18.15487214,    24.92921596,   891.43500688,
          965.83692678,  1023.16627526,    31.31001959,    28.81302863,
           23.21309011,    24.92604088]
```

```
var = [ 68.60648957,  63.97213147,  59.88279466,  13.66745704,  16.27517777,
  16.75151529,   0.55665669,  68.89530017,  59.48288375,  31.67465839,
  13.40834459,  14.62242957,  17.1366787,    0.54971325,]

with tf.Session() as sess1:
#     sess.run(tf.global_variables_initializer())
    saver.restore(sess1, dst)
    pr_dict = {x_go: a}
    prediction = sess1.run([y_go], feed_dict = pr_dict)
    pred = prediction[0].T * var[7:]
    prd = pred + mean[7:]
    print("Prediction is: ")
    print(prd[0])
```

Appendix G

Bibliography

1.Simon Batchelor (July 27th, 2015) *Will Solar Photovoltaics Increase Their Efficiency Soon? / Will Solar Photovoltaics Continue to Decrease their Cost?* Retrieved September 2017

2. Fraunhofer Press-media (January 12th, 2014) *new world record for solar cell efficiency at 46 percent* Retrieved September 2017

3. Kevin Bullis (June 20th, 2014) *Sharp Demonstrates Ultra-Efficient Solar Cells* Retrieved September 2017

4. Michael D. McGehee (September 19th, 2014) *Fast-track solar cells* volume 501, Retrieved September 2017

5. David L. Chandler (September 5th, 2013) *Solar-cell manufacturing costs: innovation could level the field* Retrieved September 2017

6. Smiti (November 23rd, 2014) *India Eyes 100 GW Solar Power Capacity By 2022* Retrieved September 2017

7. Meyer Burger (November 1st, 2017) *Increasing Efficiency of Solar Cells* Retrieved September 2017

Appendix H

Slides used for Presentation

The slides used in the final presentation are attached here.

Problem Statement 1 was repeated thrice, with 3 pairs of input:
1. Surface Concentration & Depth
2. Sheet Resistance & Depth
3. Surface Concentration & Sheet Resistance

Accuracy obtained in each of the above cases can be found on page 9 of slides.

Details of building a neural network model can be found on pages 15-17.

Methodology for future work has been discussed on slide page 19.

Update on XSolar-Hetero & Machine Learning Project:

Boron Tube Diffusion Process Optimization using Random Forest Regressor & Neural Networks

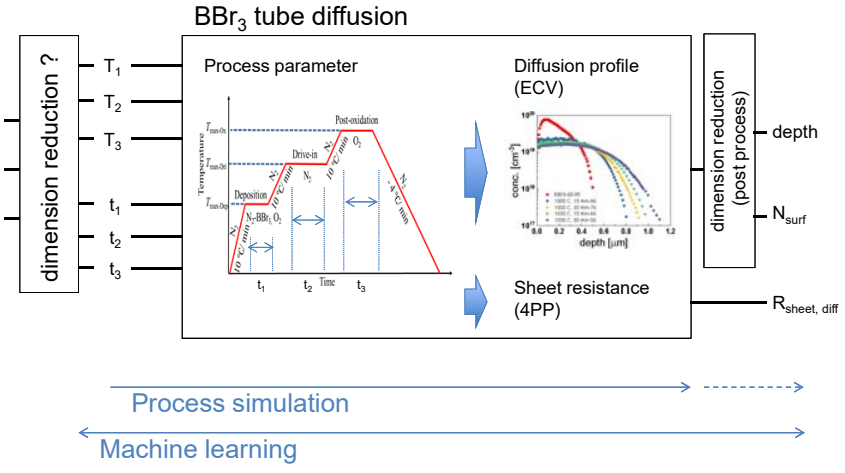
Ananya Kshatrapati Shivaditya (internship student, SERIS)
Gautam ANAND (software engineer, SERIS)

Dr Rolf STANGL (Head: NovelCellConcepts&Simulation, SERIS)
Solar Energy Research Institute of Singapore (SERIS)

SERIS is a research institute at the National University of Singapore (NUS). SERIS is sponsored by the National University of Singapore (NUS) and Singapore's National Research Foundation (NRF) through the Singapore Economic Development Board (EDB).

Level-1 learning (process simulation)

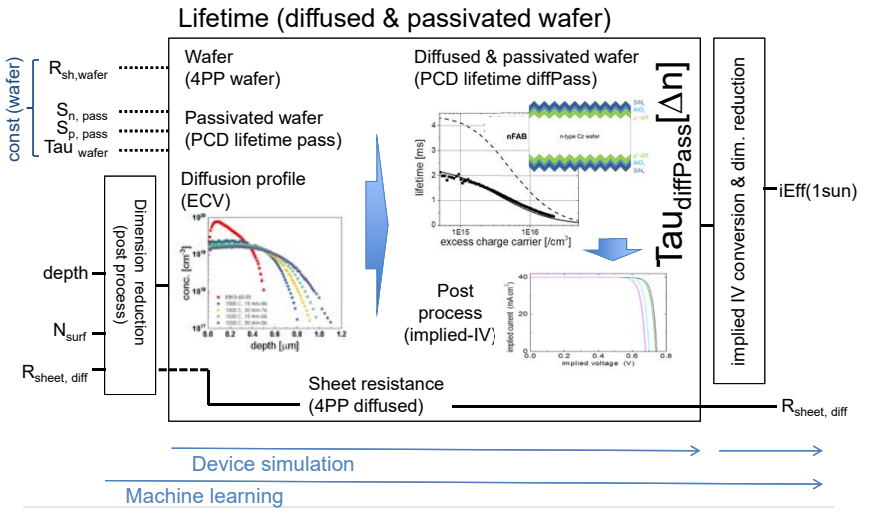
Machine Learning: Predict diffusion profile BBr_3 tube diffusion



SERIS is a research institute at the National University of Singapore (NUS). SERIS is sponsored by the National University of Singapore (NUS) and Singapore's National Research Foundation (NRF) through the Singapore Economic Development Board (EDB).

Level-1 learning (device simulation)

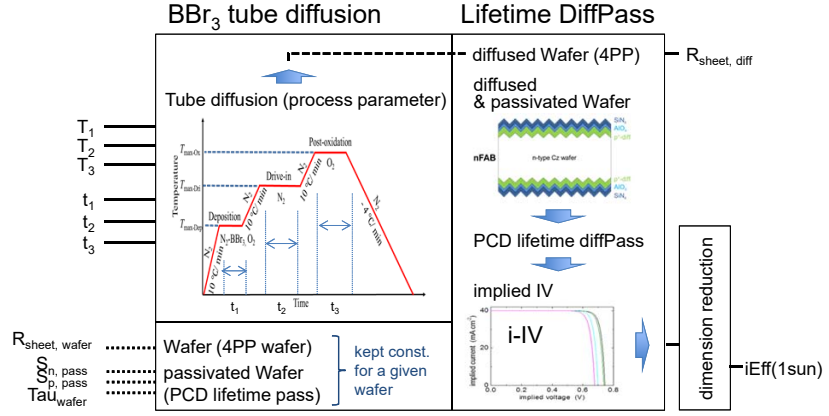
Machine learning: Predict lifetime of a diffused & passivated wafer



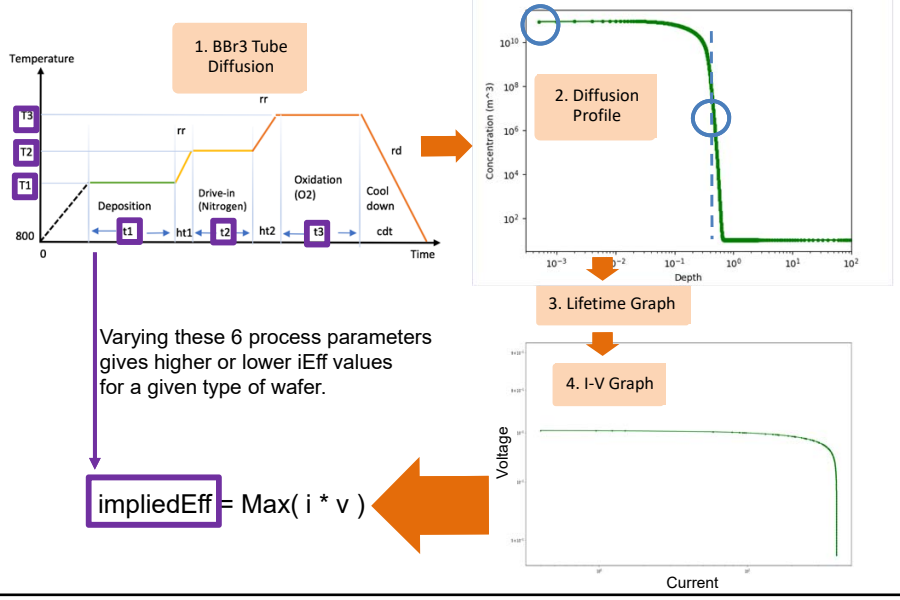
Level-2 learning (process & device sim)

Machine learning: Predict process parameters to improve iEff

(for $R_{sh,diff}$ <100 Ω/sq)



Scope of Application



Varying these 6 process parameters gives higher or lower iEff values for a given type of wafer.

impliedEff = Max(i * v)

Machine learning problem statements

Problem Statement-1:

Given certain values of surface concentration and depth, predict the set of process parameters which should be used to get that diffusion profile.

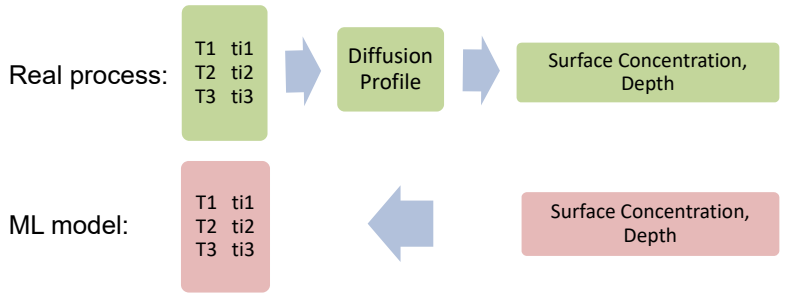
Problem Statement-2:

Given a certain set of process parameters and the corresponding implied efficiency they achieve, recommend a new set of process parameters which will increment the implied efficiency (under the constraint $R_{sheet,diff}$ <100 Ω/sq) in a small local margin.

SERIS is a research institute at the National University of Singapore (NUS). SERIS is sponsored by the National University of Singapore (NUS) and Singapore's National Research Foundation (NRF) through the Singapore Economic Development Board (EDB).

Problem Statement 1: Goal

To gain highest accuracy in reverse engineering,
i.e. obtaining values for process parameters.

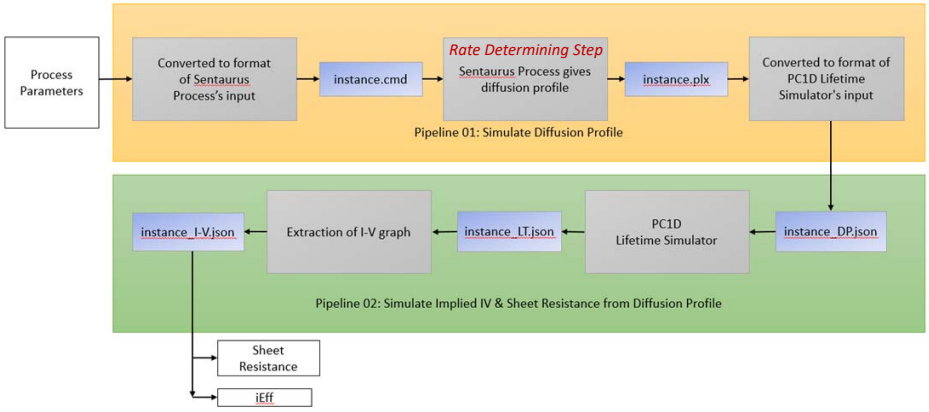


Methodology

1. Machine Learning Approach
2. Data Creation
   a. Pipeline of APIs
   b. Future retraining with real data of same shape
3. Data Quality Check
4. Model Creation
   a. Choice of algorithm: Random Forest on ScikitLearn
   b. Prediction in 1 direction (reverse)
5. Accuracy of prediction & demo
6. Future Improvements
   a. Prediction in 2 directions (forward)
   b. Comparison with other algorithms
   c. Integration into XSolar-Hetero

Data Generation Pipeline

We need data!



Range Selection

T1, T2, T3 ϵ [800 , 1050] ° C
(Constrained by T1 <= T2 <= T3)

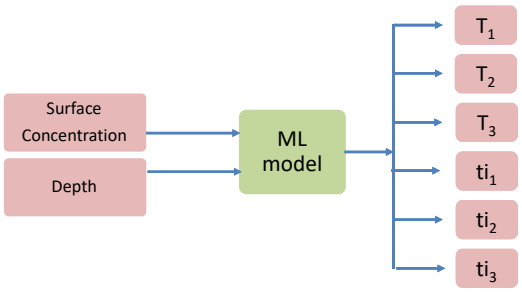
ti1, ti2, ti3 ϵ {0} $\cup$ [10, 50] minutes

Step size:
T* ≈ 40 ° C
ti* ≈ 10 minutes

Total number of points: 6255
Time taken for generation: 2 weeks
(Omitting instances with $R_{sh,diff}$ >100 Ω/sq)

Shape of Data

Problem Statement 1



Input Data	min	max	range
90%Depth (µm)	0.255	1.45	1.19
SurfConc (cm^(-3))	2.14E+18	2.36E+20	2.34E+20

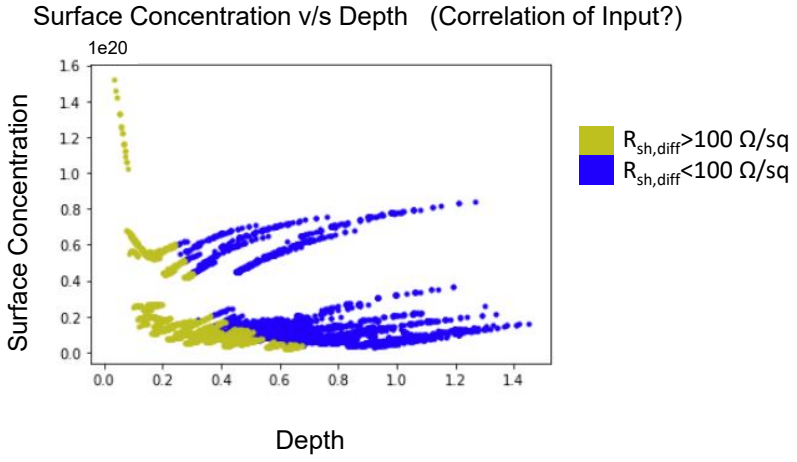
Additional Data	min	max	range
iEff (%)	23.6	25.75	2.15
SheetRes (Ohm/sq)	14.46	99.94	85.47

Output Data	min	max
T1 (°C)	800	1050
T2 (°C)	800	1050
T3 (°C)	800	1050
ti1 (mins)	10	60
ti2 (mins)	0	50
ti3 (mins)	0	50

SERIS is a research institute at the National University of Singapore (NUS). SERIS is sponsored by the National University of Singapore (NUS) and Singapore's National Research Foundation (NRF) through the Singapore Economic Development Board (EDB).

Data Quality Check

Problem Statement 1

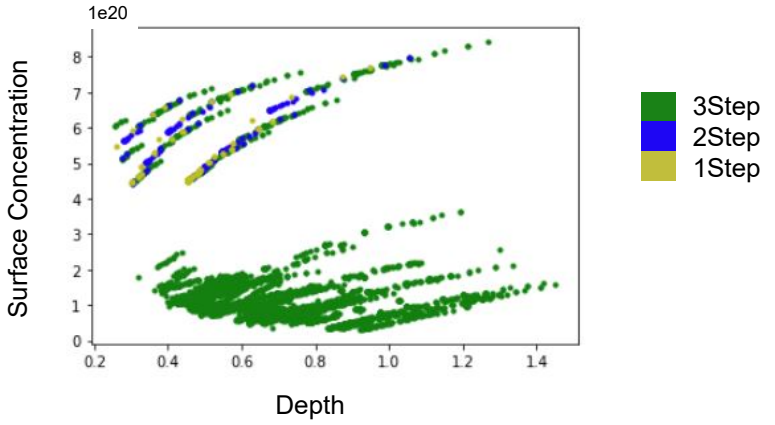


SERIS is a research institute at the National University of Singapore (NUS). SERIS is sponsored by the National University of Singapore (NUS) and Singapore's National Research Foundation (NRF) through the Singapore Economic Development Board (EDB).

Data Quality Check

Problem Statement 1

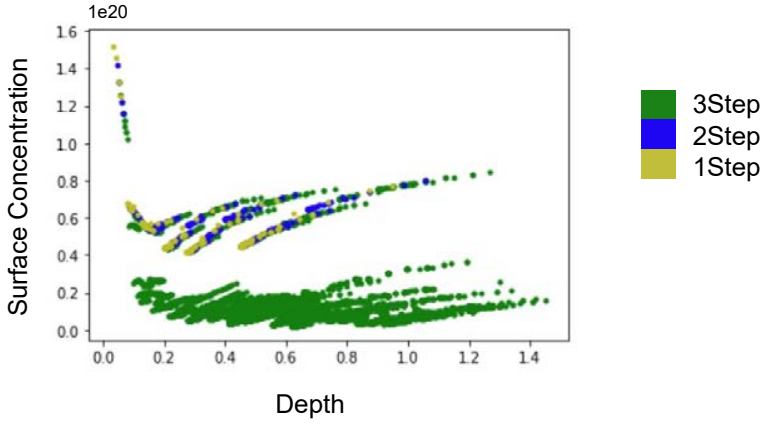
Surface Concentration v/s Depth ($R_{sh,diff}$ <100 Ω/sq)



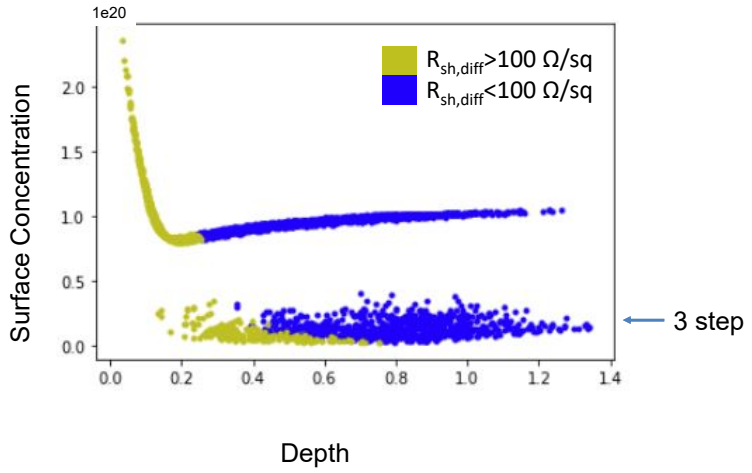
Data Quality Check

Problem Statement 1

Surface Concentration v/s Depth ($R_{sh,diff}$ without constraint)



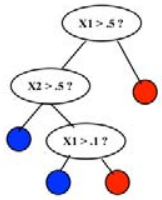
Data Quality Check (random data)

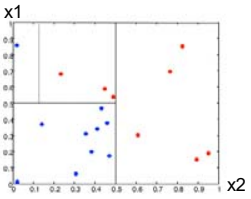


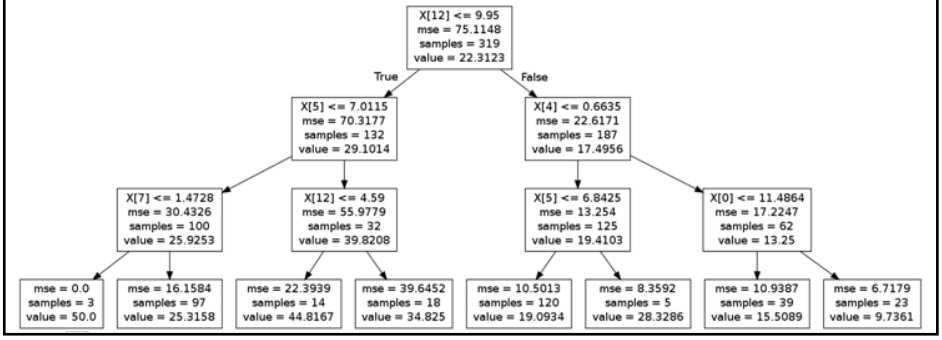
Model Creation

Random Forest Regressor

X has 12 features
mse = mean square error
value = mean value in leaf/node







Model Creation

ML model: T1, ti1, T2, ti2, T3, ti3 ← Surface Concentration, Depth

a) Learning in one direction (reverse)

Random Forest Regressor for each of the six parameters.
The poor score on time parameters was improved by:

i. Increasing no of trees in forest
ii. Using increasing feature set for the next parameter's prediction, ranked by highest individual accuracy

Accuracy of Prediction with Demo

Prediction in reverse direction (with inputs of SurfConc & Depth):

T_1: 96.99% ti_1: 87.09%
T_2: 98.74% ti_2: 96.73%
T_3: 99.65% ti_3: 99.06%

Prediction in reverse + forward direction:
1. For inputs of Depth and Surface Concentration:
Surface Concentration: 86.23%
Depth at 90% Concentration: 94.61%

When repeated with inputs of:
2. Depth and Sheet Resistance:
Sheet Resistance: 80.15%
Depth at 90% Concentration: 88.34%

3. Surface Concentration and Sheet Resistance:
Sheet Resistance: 95.74%
Surface Concentration: 85.83%

Future Work

a. Bidirectional Prediction
b. Comparison with other algorithms
c. Integration on Web Based Platform, deployment for interface is being discussed.

Problem Statement 2: Goal

Given a set of process parameters (and its corresponding implied efficiency iEff), we want a **new set of process parameters** which will give a **higher iEff**, under the constraint that the sheet resistance is <100 Ohm/sq (in order to be contactable)

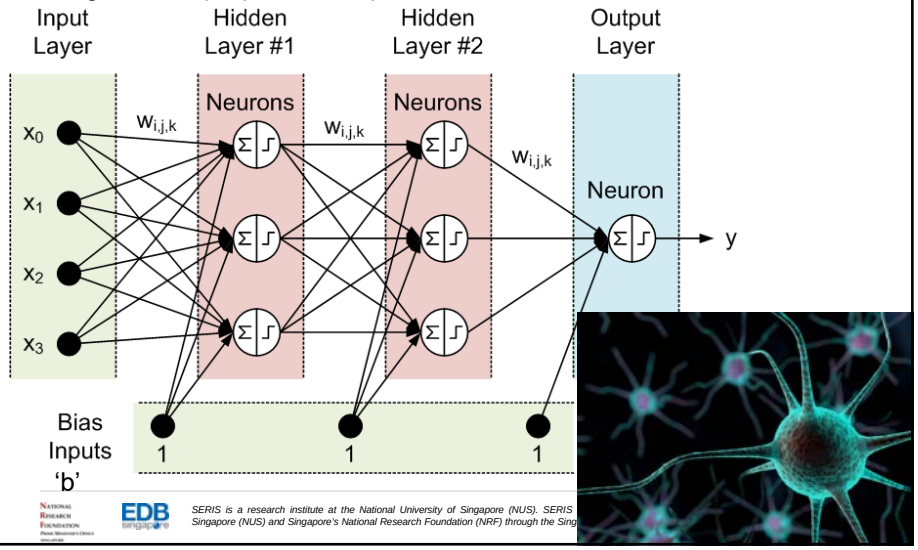
Methodology

1. Artificial Neural Network Approach
2. Data Creation
   a. Pipelining as before, with randomization.
   b. Ascending, jump
   c. Input and output data shape
3. Data Quality Check
4. Model Creation
   a. Loss Graph
   b. Learning Rate and regularization
   c. Optimizing number of nodes and layers
   d. Initialization techniques
   e. Hyperparameter search
5. Accuracy of Prediction & demo
6. Future work

NN Approach

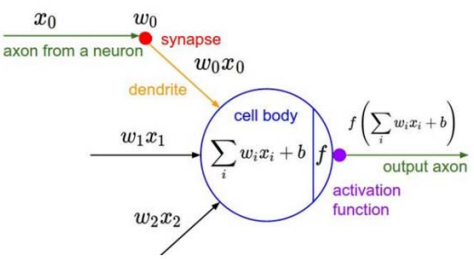
Layers, Nodes and neurons

Weights, backprop, bias, dropout

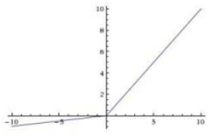


NN Approach

Activation Functions



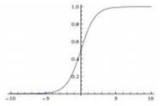
Leaky ReLU
max(0.1x, x)



(Most used)
Comparison with rule-based
Classification
"Features" in an ANN

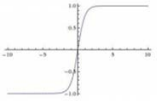
Sigmoid

$$\sigma(x) = 1/(1+e^{-x})$$

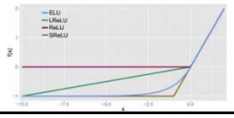


Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

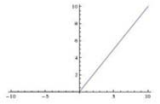
tanh tanh(x)



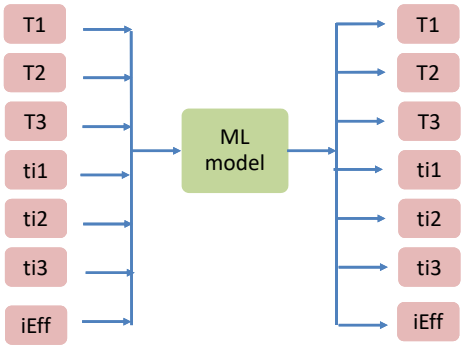
ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$



ReLU max(0,x)



Shape of input and output data



SERIS is a research institute at the National University of Singapore (NUS). SERIS is sponsored by the National University of Singapore (NUS) and Singapore's National Research Foundation (NRF) through the Singapore Economic Development Board (EDB).

Data Creation

Total data points: 9726 ≈ 10,000 points, randomly generated

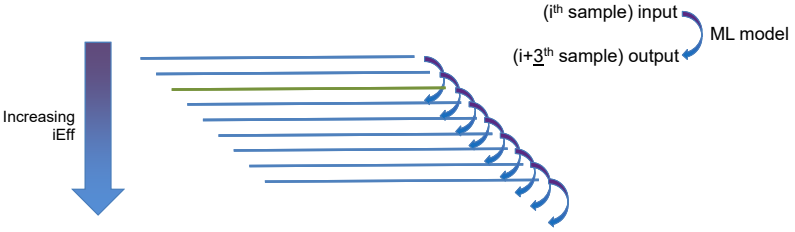
How much time: ≈ 3 Weeks

Range:

Inputs and Outputs	min	max
T1 (°C)	800	1050
T2 (°C)	800	1050
T3 (°C)	800	1050
ti1 (mins)	10	60
ti2 (mins)	0	50
ti3 (mins)	0	50
iEff (%)	23.56	26.17
SheetRes (Ohm/sq)	14.46	403.03

Dataset Generation

1. Vary the process parameters across their specified range, with random combinations, and simulate their expected iEff. The more number of instances, the better.
2. Sort them all in ascending order of iEff
3. Form pairs of i^{th} sample with $(i+3)^{th}$ Chosen by trial and comparison

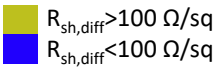


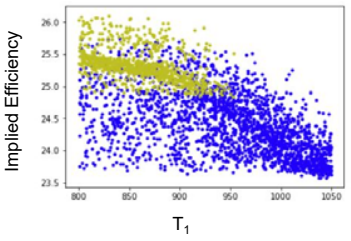
Each line is an instance of 6 process parameters & resulting iEff.

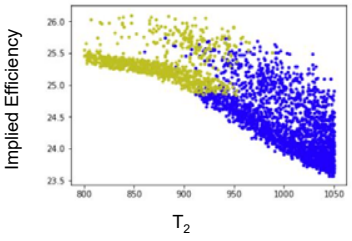
Global maximum within the dataset doesn't make sense. If given an external input, there may very well be a better output than what's in the training data.

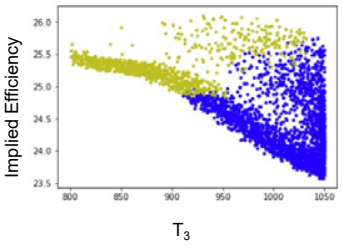
Data Quality Check

Problem Statement 2



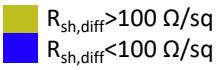


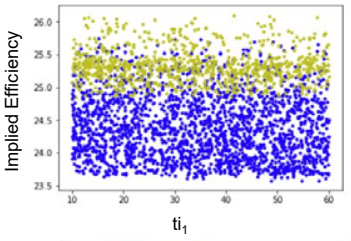


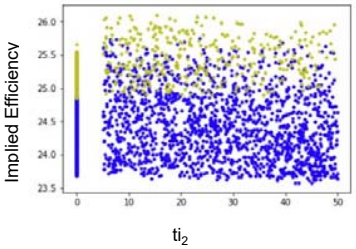


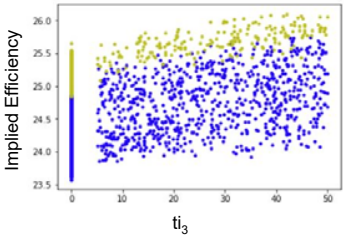
Data Quality Check

Problem Statement 2





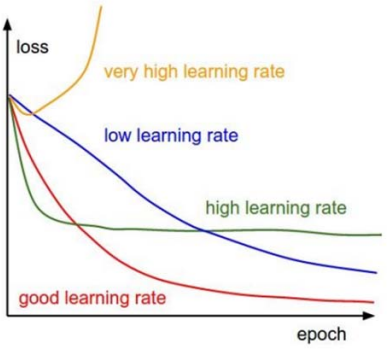


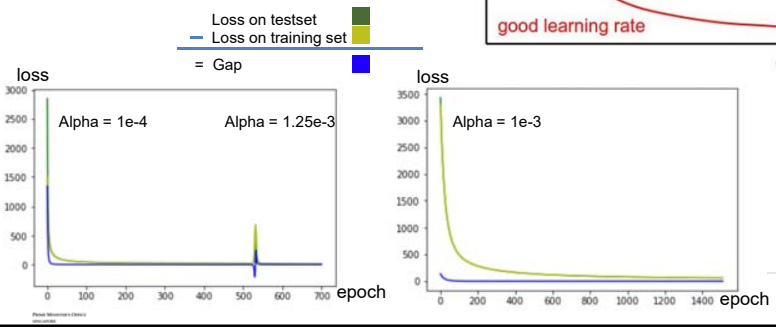


Model Creation (TensorFlow) Problem Statement 2

What's a loss graph?

1. What's the learning rate?
2. What is accuracy?
3. How many layers?
4. How many nodes in each layer?





Model Creation (TensorFlow) Problem Statement 2

Learning Rate and regularization:

If loss barely changes, the learning rate is too low.
If loss value is almost naN, the learning rate is too high.

Loss value should go down with no regularization.
i.e. Should lie somewhere in 1e-5 to 1e-3

Regularization is used to prevent overfitting.

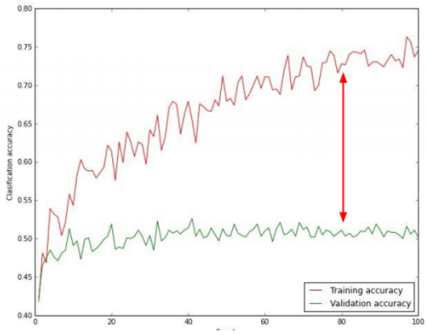
The aim is to find a learning rate that decreases loss even with small regularization.

SERIS is a research institute at the National University of Singapore (NUS). SERIS is sponsored by the National University of Singapore (NUS) and Singapore's National Research Foundation (NRF) through the Singapore Economic Development Board (EDB).

Model Creation (TensorFlow) Problem Statement 2

<u>Optimizing no of layers and nodes</u>

Monitor and visualize the accuracy:



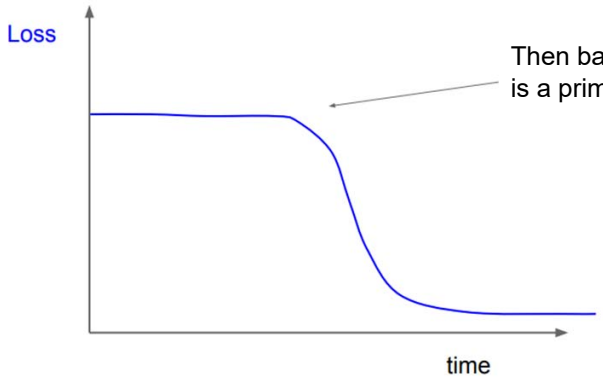
big gap = overfitting
=> increase regularization strength?

no gap
=> increase model capacity?

Model Creation (TensorFlow) Problem Statement 2

Initialization

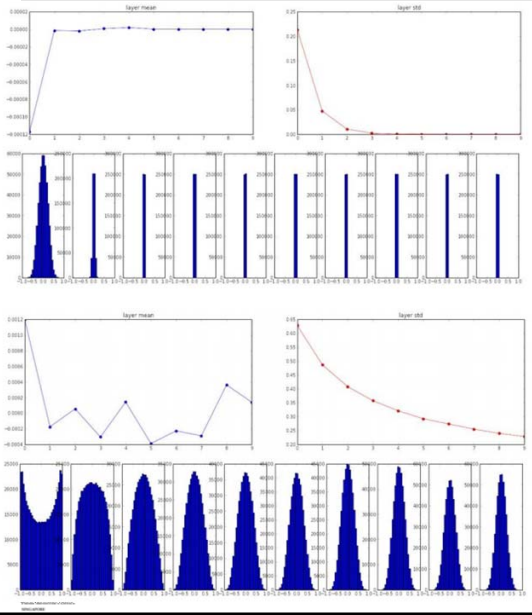
If the loss graph looks like:



Then bad initialization is a prime suspect.

Initialization Techniques

Problem Statement 2



Reasonable initialization.
(Mathematical derivation assumes linear activations)

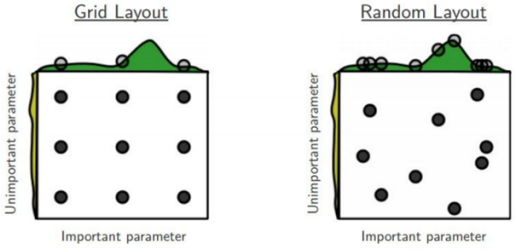
Model Creation (TensorFlow)

Problem Statement 2

Hyperparameters include:

Learning rate
Loss function
mini-batch size
epochs
momentum

Random Search vs. Grid Search



Accuracy of Prediction with Demo

Jump size
Validation with the pipeline

What next?

Other Applications:

- Separate Diffusion profiles for front and rear
- Metal contact

Stochastic variations in real data

System to incorporate real data

1. RecurrantNNs
2. Transfer Learning
3. TensorFlow Serving

Transfer learning

1. Train on large database

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end

3. If you have medium sized dataset, **"finetune"** instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

Recurrent NN

1. In their simplest form (RNNs), they are just Neural Networks with a feedback loop
2. The previous time step's hidden layer and final outputs are fed back into the network as part of the input to the next time step's hidden layers.