Capacitance Estimation using DL Model

Objective

To design a DL Algorithm that can calculate the capacitance of the wire depending on the length, width and thickness of the wire.

Methodology (Data generation & Collection)

The following steps were taken to reach the objective:

- Firstly, the data set of the different values of the length, width, thickness of the wire and their corresponding values of capacitance is prepared using **OrCAD** Software and other simulation tools by varying the values of the dimensions of wire and collecting the corresponding capacitance values.
- Our dataset is a wire capacitance table on 20nm process node. It is a 2D array of wire-length, wire-width, temperature and wire-capacitance. The values of the length, width and thickness of the wire are in metre whereas the capacitance is measured in Farads.

Wire-length	**Wire-width**	**Wire-Thickness**	**Capacitance**
(meters)	(meters)	(meters)	(Farads)
2.663e-06	9.713e-08	9.874e-10	5.201e-16
4.626e-06	1.054e-07	9.325e-10	1.023e-15
4.210e-06	8.348e-08	1.097e-09	6.546e-16
4.654e-06	1.010e-07	1.021e-09	9.115e-16
5.085e-06	1.175e-07	9.807e-10	1.183e-15
5.002e-06	6.509e-08	9.565e-10	7.140e-16
4.970e-06	8.999e-08	9.861e-10	9.091e-16
6.115e-06	1.030e-07	1.046e-09	1.191e-15
6.483e-06	1.095e-07	9.673e-10	1.435e-15
5.444e-06	1.336e-07	9.952e-10	1.402e-15

Figure 1: A subset of the generated Dataset

- Since our length, width and thickness values are small (ranging from 0.1nm to 10µm) and capacitance values are even smaller (ranging from 0.2 fF to over 2fF), it is very important to normalize the dataset right after loading.

```
data_mean = np.mean(dataset, axis=0)
data_std  = np.std(dataset, axis=0)

def normalize(d, mean, std):
    return (d - mean) / std

dataset = normalize(dataset, data_mean, data_std)
np.random.shuffle(dataset)
```

- To load the training data first we import necessary libraries like Panda and Sklearn.
- For training the model we imported from Keras the Dense and LeakyRelu layers.

```
import pandas
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LeakyReLU
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

- We split the data set in X_train, X_test, y_train and y_test with 98:2 ratio.

```
x_train,x_test,y_train,y_test = train_test_split(data,target,test_size=0.02,random_state=4)
```

- Then we load sequential model and added the three dense layers with the following parameters.

```
model=Sequential()

model.add(Dense(15,input_shape=(3,)))
model.add(LeakyReLU(alpha=0.05))
model.add(Dense(10))
model.add(LeakyReLU(alpha=0.05))
model.add(Dense(1))
model.add(LeakyReLU(alpha=0.05))
model.compile(optimizer='SGD',loss='mse',metrics=['acc'])
```

- Stochastic Gradient Descent was used as the optimizer.

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 15)                60
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 15)                0
_________________________________________________________________
dense_16 (Dense)             (None, 10)                160
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 10)                0
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 11
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 1)                 0
=================================================================
Total params: 231
Trainable params: 231
Non-trainable params: 0
```

Figure 2: Model Summary

Output Results

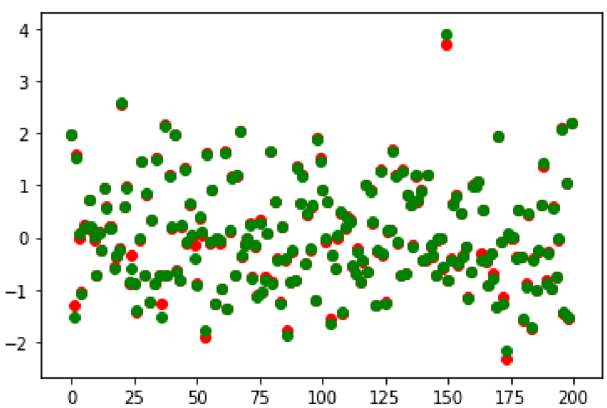

Figure 3: Predicted Values for the Test Dataset

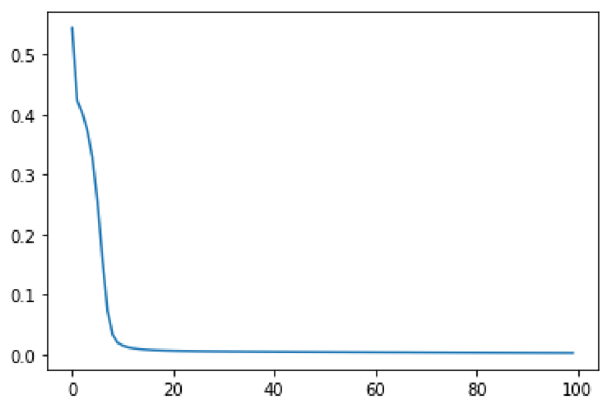

Figure 4: Loss Variation vs Number of Epochs