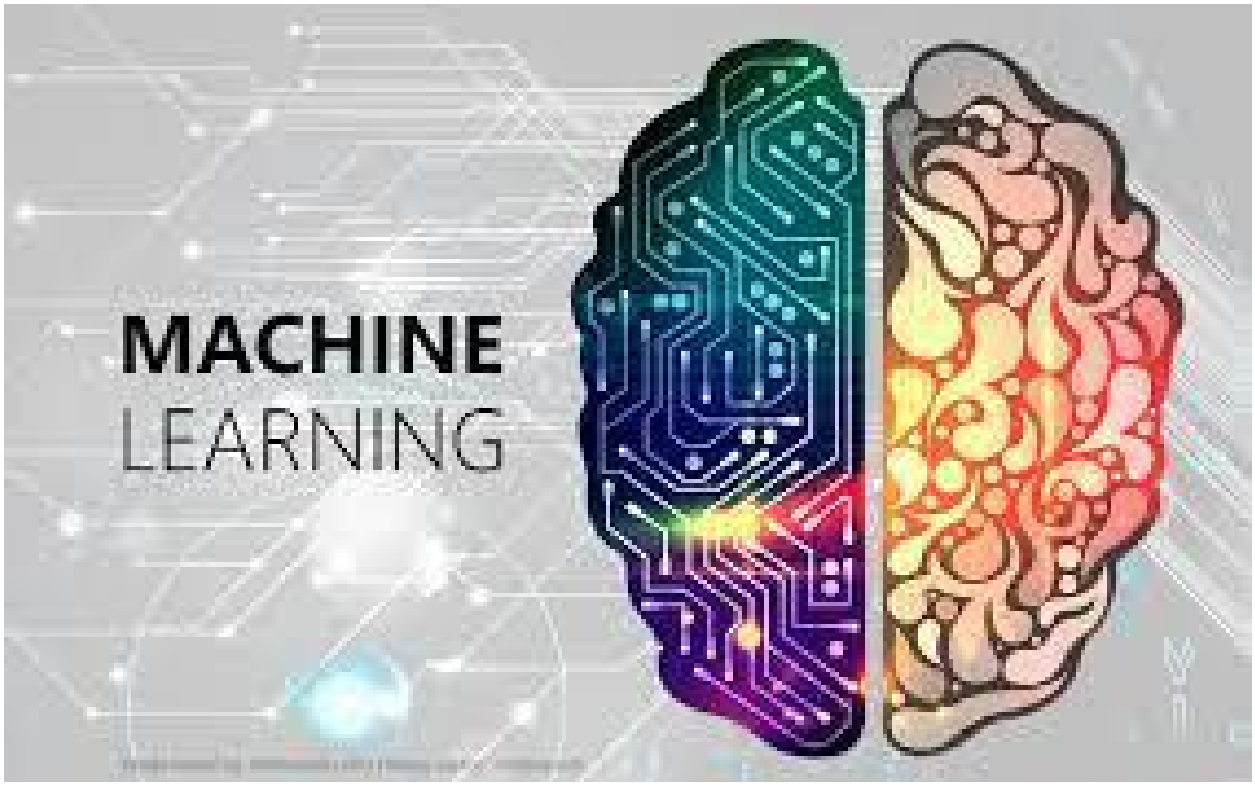
Machine Learning Project

RUL (Capacity) Prediction of Lithium-Ion-Batteries



Team Members Group 19:

- Nityansh Doshi (191080053)
- Shubham Pawar (191080073)
- Sahil Satpute (191080066)
- Shaikh Mohammed Ammar (191080071)
- Atharva Bhosale (191080006)

Problem Statement:

- (Neural Network with SGD Optimizer) This is a project to predict the remaining useful life of lithium-ion-batteries and compare which algorithm performs better (whether it's a neural network with stochastic gradient descent **or** Multiple Linear/ Polynomial Regression). The project was primarily to predict the capacity of the battery.

DataSet: NASA Lithium-ion Battery Dataset

- The dataset is from NASA Ames Prognostics Center of Excellence . To simulate the dynamic operation conditions in real applications, 18650 LiCoO2 batteries (2.1 Ah) were cycled under a series of random currents rather than the constant discharge currents.
- Each loading period lasted for 5 minutes. A 2 A charging and discharging test was performed after every 1500 periods (about 5 days) to measure the battery capacity.
- For our study, we used the data from B0005,B0006,B0007 and B0018 batteries. The failure thresholds for those batteries are considered as the capacities at the end of the test. The capacities are plotted against the test time (days). The capacity curve is highly dynamic and nonlinear.

Data Description:

● **Number Of Battery Samples:** 4
● **No. of charge,discharge,impedance cycles for each battery :** 616
● **Parameters considered for charge cycle :** Voltage, current,temperature,current charge,voltage charge and time
● **Parameters considered for discharge cycle:** Voltage, current, temperature, current load,voltage load ,capacity and time.

Motivation:

- This project was highly researched and the code was easy to implement.
- Furthermore, the prediction of remaining useful life is more useful in the real-life and hence it serves a real purpose.
- Hence our group was more attracted towards the implementation of this project.

Methodology:

Predicting the Remaining Useful Life (RUL) of Li-ion batteries:

- As per the proposed methodology the RUL(remaining number of charging and discharging cycles) of the Li-ion batteries are predicted using six parameters - number of cycles of battery, time measured, voltage measured, current measured, temperature measured and its known capacity.
- The extrapolation techniques for neural networks have been used by giving input as cycle number, time measured, voltage measured, current measured and temperature measured of the given battery dataset.

- Using this method, capacities for a particular number of cycles beyond this cycle are predicted.
- When the capacity of particular cycle reaches the threshold capacity value for the given battery then the RUL for that battery for the given cycle number is predicted as the difference between the current cycle number for which threshold capacity is reached and the input cycle number i.e.

Predicted RUL = Threshold cycle number

Current input cycle number:

- For example, if the battery is currently at cycle number 10, then we will start predicting the capacities for further cycles until the threshold capacity is reached.
- Let's say that the threshold capacity is reached at the 20th cycle of the battery, then at present the battery has 20 - 10 = 10 cycles

Remaining useful life:

- The real RUL is calculated as the difference of the total number of life cycles a battery has and the current cycle number at which the RUL has to be predicted.

Real RUL = Total no. of cycles - Current input cycle number

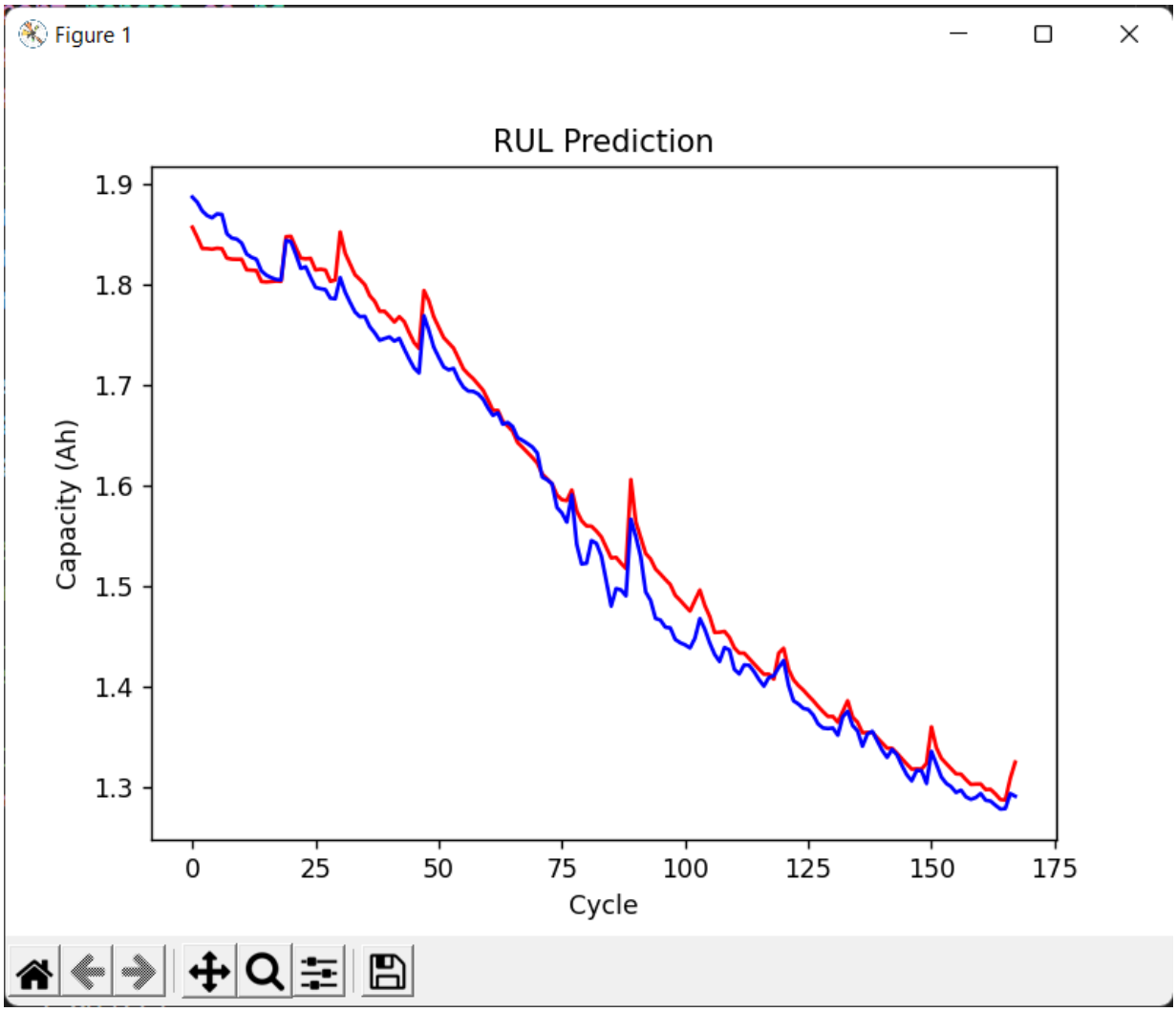
- For example, if the total number of cycles for a battery sample is 600 and we want to calculate RUL(i.e remaining number of charging and discharging cycles) for the 400th cycle then the real RUL is 600-400= 200.
- The RUL of testing battery i.e B0005 is predicted from capacity and cycle number.

- The prediction results are based on NN methods and show the desirable properties, that is the prediction curves can converge to the real capacity curves and the RUL pdfs become narrow as the time of prediction advances.
- The NN method tracks the aging variation well when collecting the data after sudden changes (100 days). This is because by adjusting the
- Model parameters using more capacity data, the NN model can effectively track the degradation trend and accordingly, achieve good prediction results.
- However, the capacity prediction curves obtained by the EXP model are obviously different from the real ones for all prediction times. The reason is that the degradation characteristic for this battery is relatively complex with strong dynamics.
- Thus, it is difficult to be tracked and predicted using the simple EXP model.

Graphs and Inferences:

RUL Prediction using SGD optimizer
Details of Neural Network

First Hidden layer: 10 neurons, Activation function= ReLu
Second Hidden layer: 5 neurons, Activation function= tanh
Input to the neural network : Cycle No.,
Output of the network : Capacity
Optimizer : SGD
Loss: Mean squared error

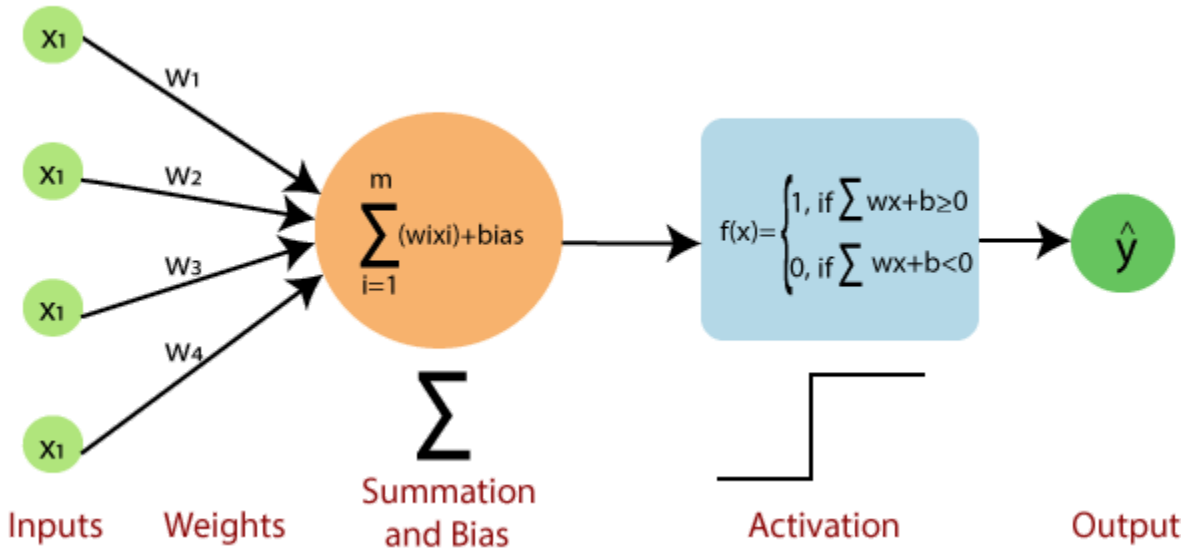


- The Red line indicates capacity values for the B0005 battery.
- The Blue line indicates the predicted values using NN (optimizer=sgd).
- The Neural network was trained on Batteries B0006, B0007, B0018 from the dataset provided by NASA
- However it is observed from the trails performed during implementation that prediction based on only cycle number does not give accurate results

Average training accuracy: 0.985
Average testing accuracy: 0.983

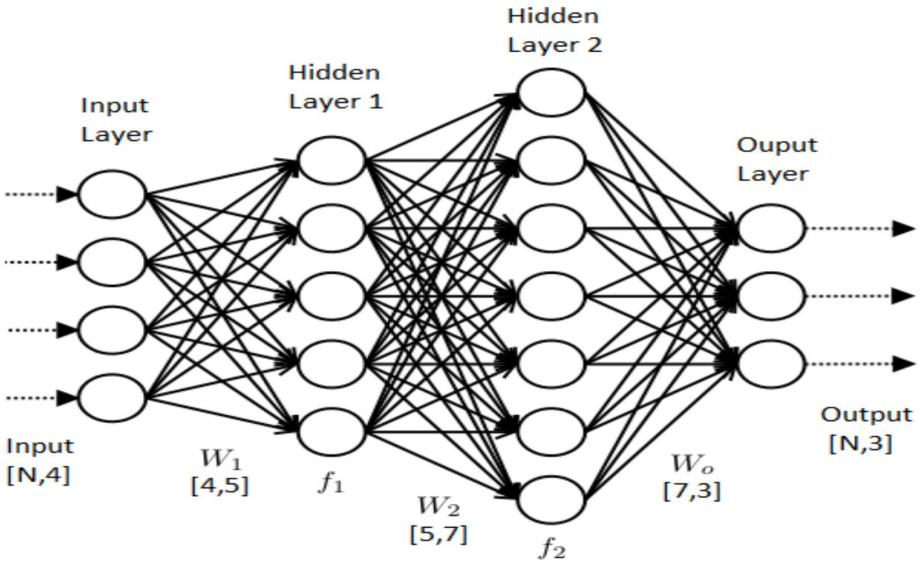
Implementation Details:

Libraries Used: Keras, tensorflow, scikit-learn, pandas, numpy, matplotlib.



Here we have used two hidden layers of Neurons with 10 in first with relu as activation, 5 in second and tanh as activation and and 1 in the output layer with Linear activation.

The structure of the Network is (5, 10, 5, 1).



SGD Optimizer:

We also added the SGD optimizers for momentum gain.

RUL Prediction through Polynomial Regression:

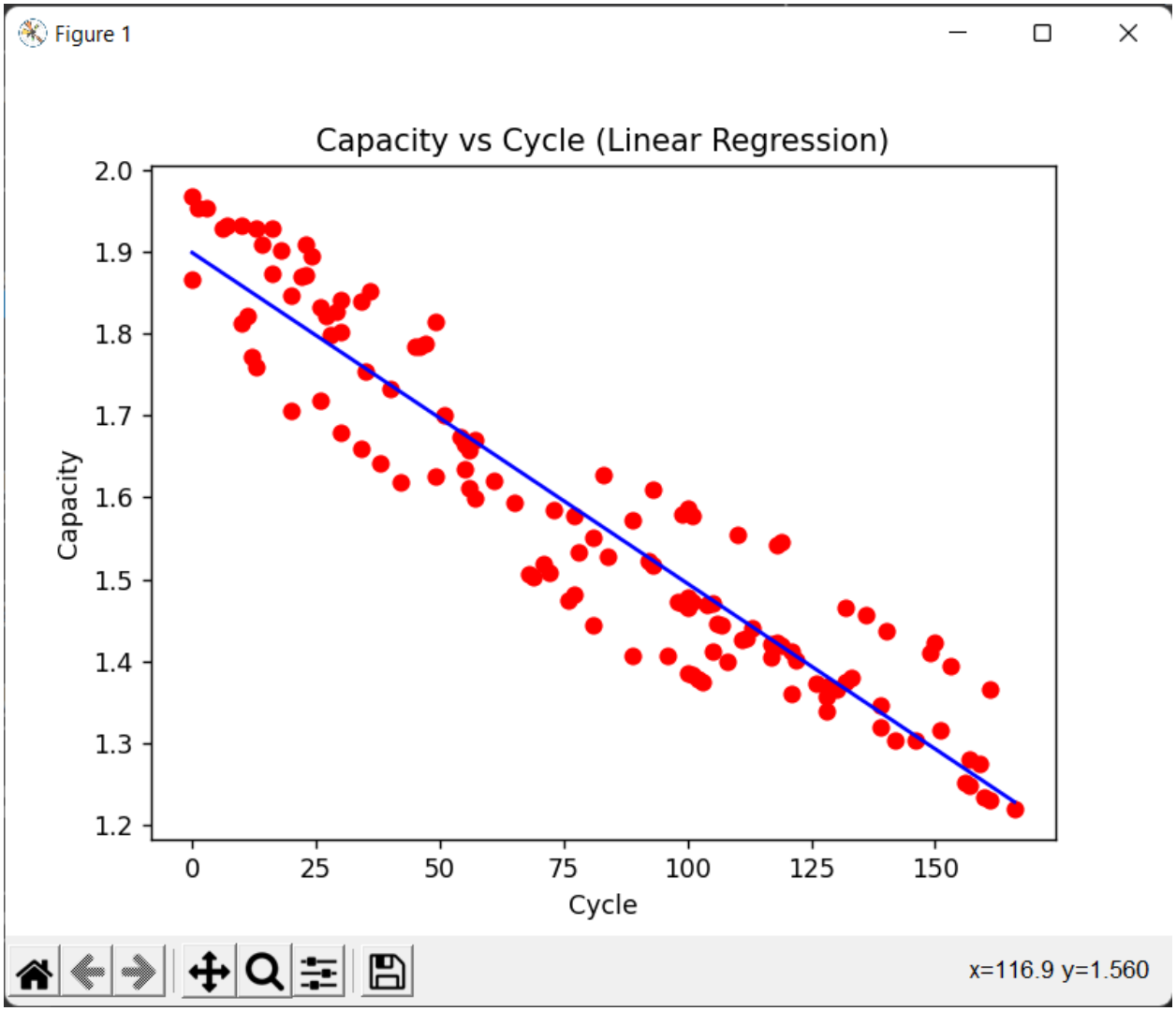
Polynomial regression is a form of Linear regression where only due to the Non-linear relationship between dependent and independent variables we add some polynomial terms to linear regression to convert it into Polynomial regression.

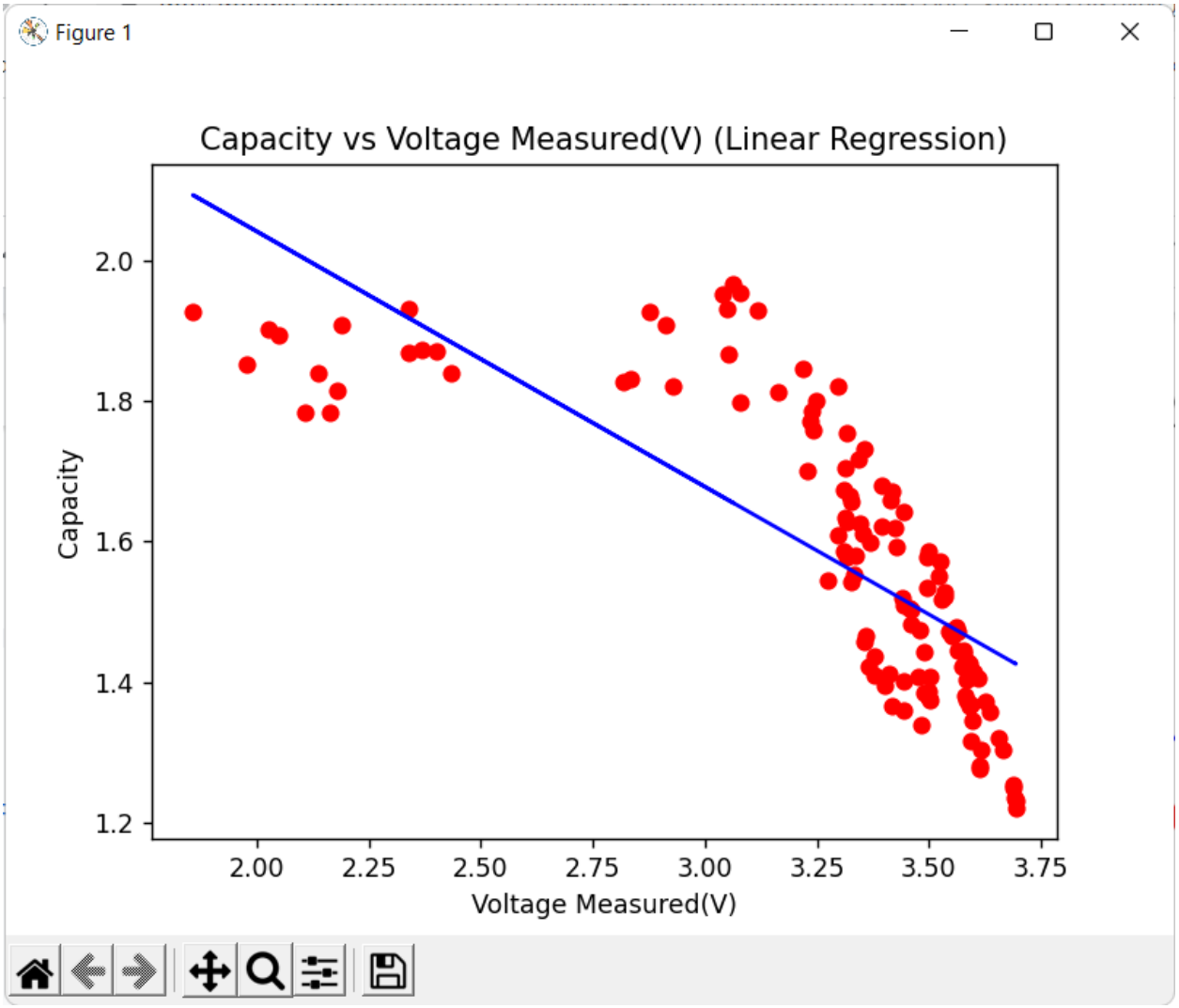
The equation of the polynomial is:

$$Y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + ... + \theta_n x^n$$

Implementation Details:
We tried for degree as 5 and we also measure the RMSE for the output which comes out to be

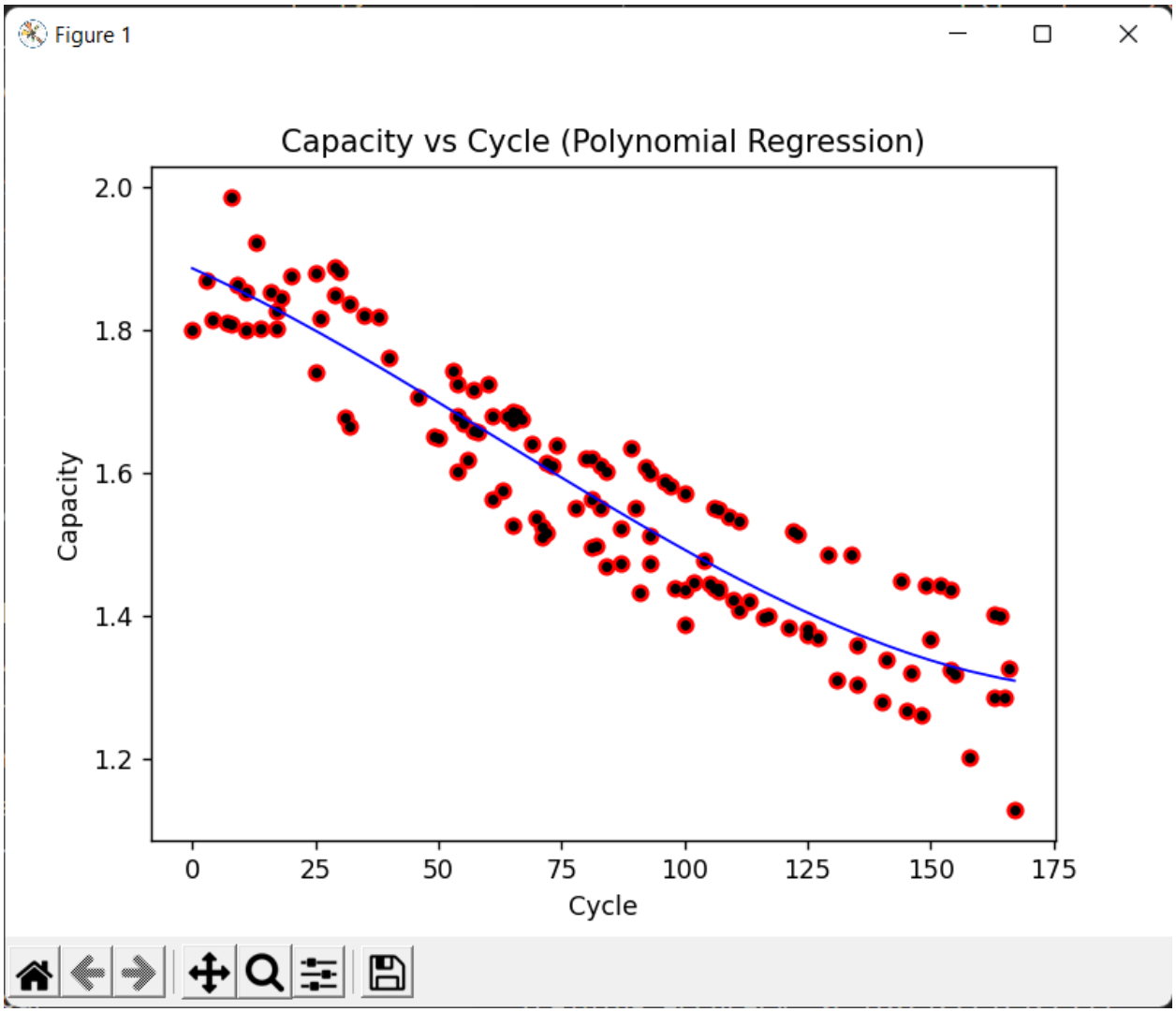
Linear regression

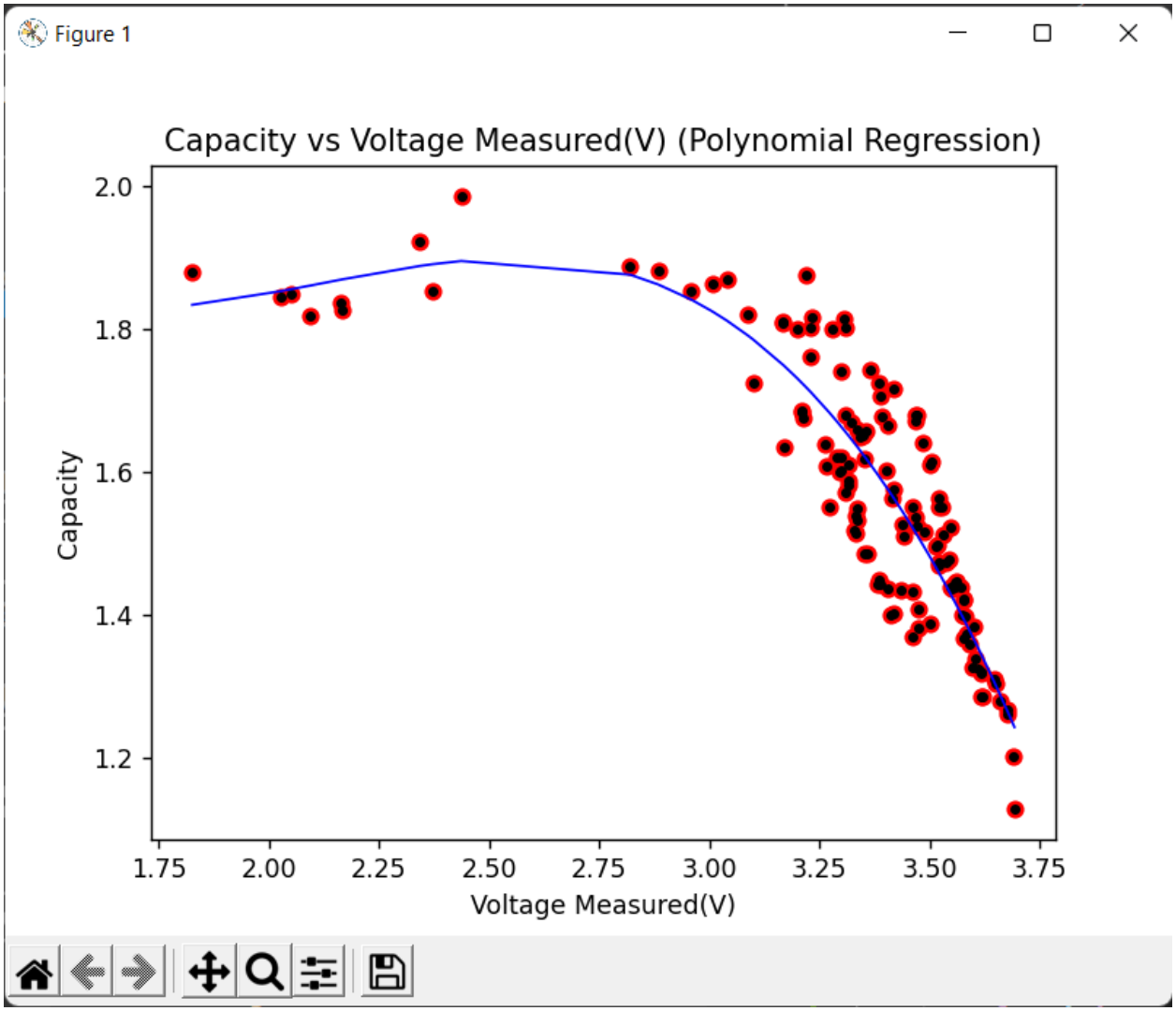




Average training accuracy: 0.978
Average testing accuracy: 0.977
Average training root mean squared error: 0.0290
Average testing root mean squared error: 0.0293

Polynomial Regression



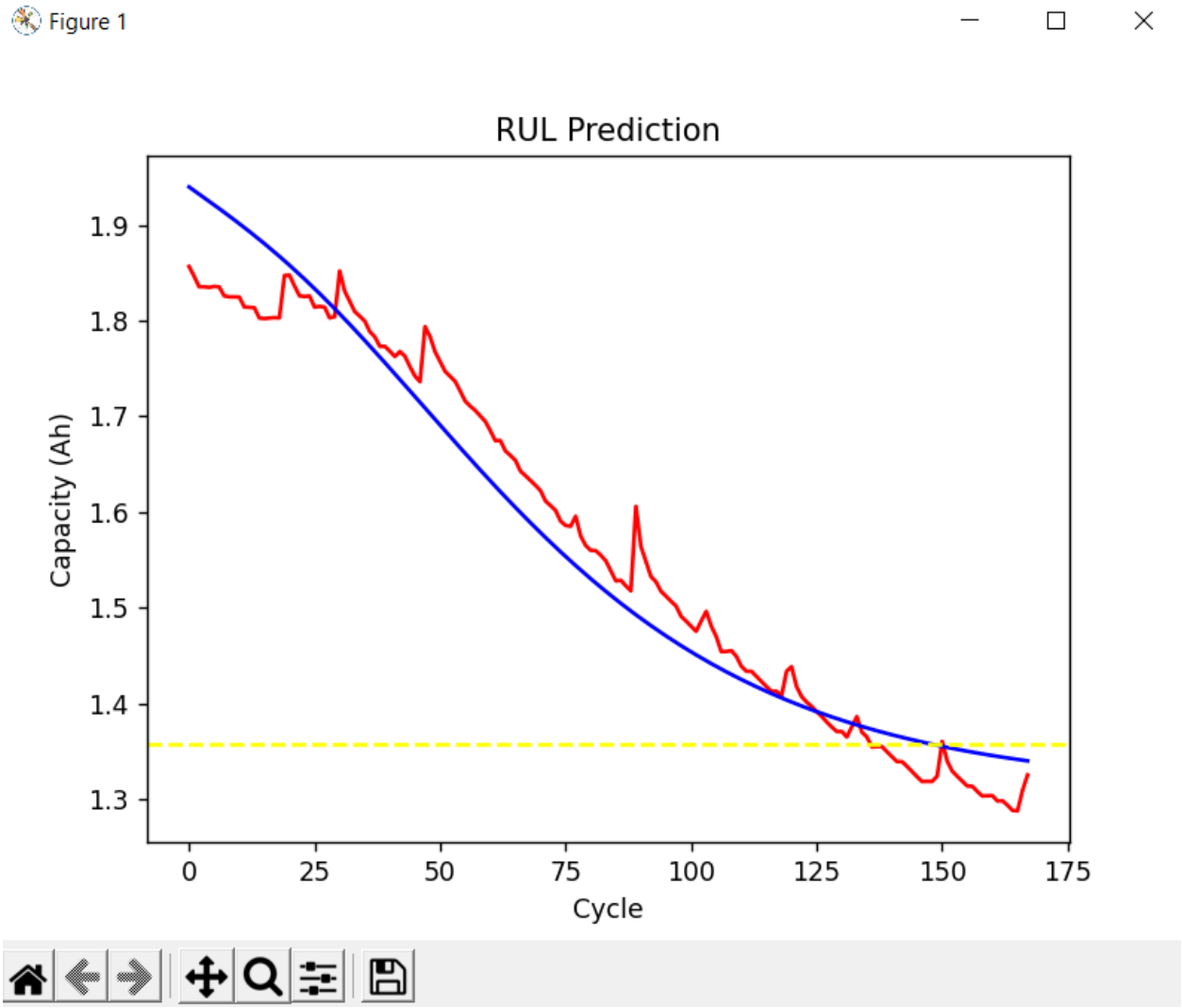


Average training accuracy: 0.996
Average testing accuracy: 0.987
Average training root mean squared error: 0.0107
Average testing root mean squared error: 0.0192

New findings:

Failed Attempts

An attempt was made to predict RUL using NN for only Input cycle number. We also tried to use an optimizer (Stochastic gradient descent).
The value of R2 must tend towards 1.00 and the value of RMSE must tend toward 0 for good accuracy. Since the accuracy was very poor it was a failed attempt.



CONTRIBUTIONS:

We implemented 5-fold cross-validation for the dataset. We also modified the code for the old libraries and removed bugs.

In the NN for considering all input parameters we changed the hyperparameters and initial values of Weights and Biases so that the models could fit better and give better accuracy for testing the dataset.

We implemented polynomial regression which was not implemented in the original research paper and got decent accuracy.

PSEUDO CODE WITH OUTPUT:

NN with All Parameters:

```
Import libraries from keras.

Take a Sequential Model from keras libraries.

Add the Neural Network Dense Layer with 5 input nodes and 10 output nodes, with tanh activation function.

Add the Neural Network Dense Layer with 5 output nodes and 10 input nodes, with tanh activation.

Add the output layer with 1 output node and 5 input nodes and with activation as Linear function.

Compile the model with SGD optimizer and also with mean_absolute_error accuracy

Fitting the ANN in the Training set

Predicting the Test set result.

Import matplotlib

Predict for each instance in testing data.

Plot the Train and testing data predictions
```

Average training accuracy: 0.985
Average testing accuracy: 0.983

Polynomial Regression:

```
Import libraries

Load the dataset


Apply stratified K_fold cross-validation

/// Linear Regression Example

for train_set and test_set in stratified K-fold split:

    Take a LinearRegression Model and Fit it to training data.

            Fit on the Training Data

            Predict for the testing data

            Measure the R2_score on testing data.

    Measure the train and test score.

            Plot a scatter plot for X_train and Y_train

            Plot a line plot for X_test[“Cycle”] and Y_predicted.

    Print the training and testing accuracy.


Print The average accuracy.

/// Polynomial Regression Example.

for train_set and test_set in stratified K-fold split:

            Take a Polynomial Regression Model and Fit it to training data with degree ____

            Fit the training data on the model.

        Fit the testing data on the model.

            Measure r2_score for predictions.
```

```
        Plot a scatter plot for X_train and Y_train

        Plot a line plot for X_test[“Cycle”] and Y_predicted

        Print the training and testing accuracy.

Print the average training and testing accuracy.
```

Average training accuracy: 0.996
Average testing accuracy: 0.987

DISCUSSION:

Performance Metrics	NN_SGD	NN_SGDWithAllParameters	Polynomial Regression
R2_Score	0.961	0.983	0.987
RMSE	0.0375	0.0251	0.0192

Polynomial Regression:

Capacity prediction:

Actual Capacity	Predicted Capacity
2.035338	1.934727
2.013326	1.986066
2.000528	2.027315

Neural Networks with all parameter:

Capacity prediction:

Actual Capacity	Predicted Capacity
1.856487	1.894513
1.846327	1.888138
1.835349	1.877921

- As NN_SGD takes one attribute as input, it does not predict as good enough in comparison with NNwithAllParameters and Polynomial Regression.
- Although NNwithAllParameters and Polynomial regression have almost similar prediction accuracy, polynomial regression works better on continuous data so it performs slightly better than NNwithAllParameters.

CONCLUSION:

1. Increasing no. of epochs does not necessarily improve accuracy.
2. Optimizers work in combination with loss function, a particular loss function may prove good for a given optimizer and not for others.
3. While dealing with multiple inputs (ex cycle no, voltage, current, temperature etc), scaling the features are important.
4. Increasing the number of layers and breadth of the neural network increases the accuracy of our predicted model
5. K-fold may increase the accuracy when the data is collected from non-uniform sources, as it addresses different parts in each split.