Project Report

Sameera Suhail

6/12/2019

Artificial Neural Network Based Modelling of Organic Photovoltaic Cells

Major Research Project
EEE80018

Sameera Suhail
Student ID: 101739677

Project Supervisor
Dr. Jagdish Patra

Contents

Abstract

Even though Organic Photovoltaic Cells (OPV) have been around since 1980s, their use was limited due to their low efficiencies. With these efficiencies now on the rise, they have become commercially relevant. The present study is based on OPVs that have poly(3,4-ethylenedioxythiophene):poly(styrenesulfonate) (PEDOT:PSS) as a hole extracting layer (HEL) and anionic fluorinated ionomers (PFI) as additives in the HEL. Analytical modeling of solar cell characteristics is difficult and to some extent inaccurate as many assumptions have to be made. We, therefore, attempt to use Artificial Neural Networks (ANN) in the case of OPVs with PFI in PEDOT:PSS layer to model two characteristics of OPVs: one optical and one electrical. Use of multilayer perceptrons along with functional link neural networks allows for comparison of both ANNs in terms of both training times and accuracies. The accuracies obtained with both MLP and FLANN are close to each other but, as expected, FLANN trains faster.

1 Introduction

In recent years, organic photovoltaics (OPVs) have become popular due to their low cost, easy manufacturing, high flexibility and high absorption coefficients. They don't require high-temperature vacuum processes for manufacturing which makes it possible to produce them in bulk, using methods such as spin-coating, ink-jet printing, spray deposition and roll-to-roll processing [1]–[12]. Despite the many benefits of adopting OPVs commercially, their use remains limited due to their low efficiency and stability. Till 2018, OPVs with efficiencies greater than 10% have been reported[13]–[17]. The degradation mechanisms responsible for low lifetimes of OPVs are environmental factors such as exposure to O_2 and H_2O, photochemical and metal ion diffusion from the electrodes.

The functioning of an organic solar cell includes a number of steps. Firstly, under illumination, a photon is absorbed by the active layer leading to the formation of an electron-hole pair. The pair (exciton) is then separated and the electrons go towards cathode and the holes go towards anode. This means that the cathode is supposed to have lower work functions to attract the electrons while the anode is supposed to have higher work functions to attract the holes. The current, therefore, flows out of the anode side of the photovoltaic cell and into the cathode. Ideally, the contact resistance at the electrode interfaces determines the device photovoltage. The smaller the resistance, the smaller is the barrier that the charges experience at the electrodes. Additional layers are included between the active layer and each electrode to modify their work functions. LiF is a common material that reduces the work function of the cathode by being used between the metal cathode and the active layer [18]. Similarly, poly(3,4-ethylenedioxythiophene):poly(styrenesulfonate) (PEDOT:PSS) increases the work function of the anode side of the cell and helps in reducing the injection barrier experienced by the holes[19].

Therefore, the PEDOT:PSS layer sitting between the Titanium Indium oxide anode and the active layer is called a Hole Extraction Layer (HEL). It has applications in other organic optoelectronic devices and is cheap, environmentally friendly and easily manufactured without requiring high temperature processing. The conducting polymer has large ionization potential, high conductivity and large transparency. The transparency ensures that light reaches the active layer where the disassociation of excitons actually occurs.

PEDOT:PSS, however, has its own share of issues. Its high chemical reactivity makes the cell susceptible to degradation due to environmental factors [20]. Its electronic properties are modified for the worse when the hygroscopic layer absorbs moisture and its work function reduces towards vacuum, thereby, decreasing the charge extraction capacity of the OPV. In the study by Howells et al. [21] anionic fluorinated ionomers (PFI) were used as additives in the PEDOT:PSS layer. The advantage attained from doing so was twofold: first to

reduce degradation of OPV from H_2O and O_2; second to tune its work function to higher values leading to better charge extraction. Different concentrations of PFI were experimented with in the study at hand [21]. Although, the effect of PFI concentration was studied on a number of OPV characteristics, in this study I attempted to use Neural Networks to model only the transmittance as a function of light wavelength and the J-V characteristics at 0 hour of continuous AM1.5 G illumination at 100mW cm-2. The rest of the report has been divided into a number of sections. Section 2 gives an overview of the development of OPVs since their discovery and the use of neural networks in past researches involving modeling of solar cell characteristics. Section 3 describes briefly the two different types of ANN that will be employed in this study and the design of experiments. Section 4 shows the analysis and results obtained. Finally section 5 gives the concluding remarks on the results and implications.

2 Literature Review

The very first OPVs consisted of a single organic layer between two metal electrodes having different work functions[22], [23]. Then in 1986, heterojunction bilayers came to the scene where two layers having n-type and p-type properties respectively were sandwiched between the two electrodes [24]. As more research led to the development of conjugated polymers, OPVs with polymer-fullerene bulk heterojunction became popular [25], [26]. The most widely used one is P3HT:PC61BM bulk heterojunction, where PC61BM is a fullerene derivative acceptor with high electron affinity and P3HT is a polymer donor with high ionization potential. The aim of bulk heterojunction is to intermix the acceptor and donor polymers in bulk in order to increase the area for charge separation to occur. This ensures that recombination stays to a minimum and all excitons are disassociated. The aforementioned P3HT:PC61BM devices are made using solution casting of P3HT and PC61BM blends [27]. Attempts have been made to accurately model the characteristics of OPVs in order to enhance the design of solar cells. Suliman et al. [28] achieved optimal device performance using response surface methodology using three variables: polymer concentration, polymer-fullerene ratio and active layer spinning speed. The use of a statistical tools meant the use of complex mathematics. Pillai et al. [29] opted to use simple circuit equations of a reverse double diode model to model the behavior of nav-100 an OPV. The unknown parameters in the double diode model were nevertheless extracted using Genetic Algorithm. Fallahpour et al. [30] adopted Transfer Matrix Method and Drift Diffusion concept to model the model light absorption and charge transport respectively for energetically disordered solar cells. In such studies involving analytical modeling techniques, results are often obtained after several assumptions and approximations [31].

Artificial neural networks (ANNs) have, however, been known to give excellent results in areas involving high noise and non-linear input to output mapping [32]. The present study is based on the experimental study

by Howell et al. [21], where the optoelectronic and topographical properties of PEDOT:PSS with different concentrations of PFI additive were studied. The characteristics studied were transmittance, topography, work function, contact angle with water, J-V device characteristics and degradation studies using Jsc, Voc, Fill Factor (FF), and Power Conversion Efficiency (PCE) as parameters [21].

3 Research Design

3.1 Experiment Characterization and Input Dataset

The transmission (T%) vs. wavelength (λ) and current density (J) vs. voltage (V) characteristics were modeled using ANNS. Both characteristics were first modeled using multilayer perceptrons (MLPs) and then functional link neural networks (FLANNs). To decide the best architecture, the number of hidden layers in MLP was restricted to one as the number of features was two at most. The number of neurons in the hidden layer was, however, varied to arrive at the architecture giving the best performance. With regards to FLANN, the polynomial used for functional expansion was varied among Chebyshev, Legendre, Trigonometric and Power. The polynomial with relatively better performance among these four was finally chosen.

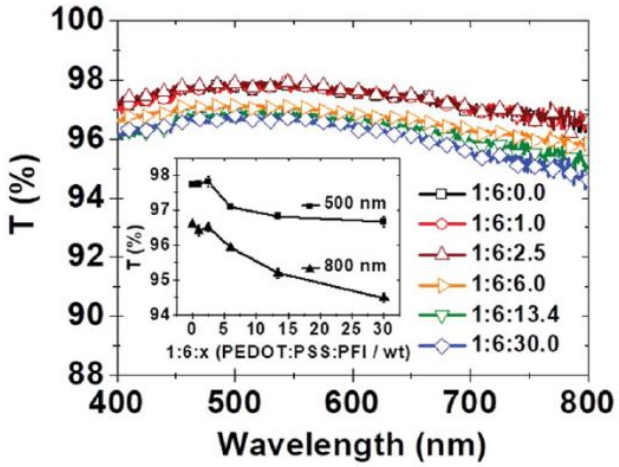

Figure 1: Transmittance vs. wavelength

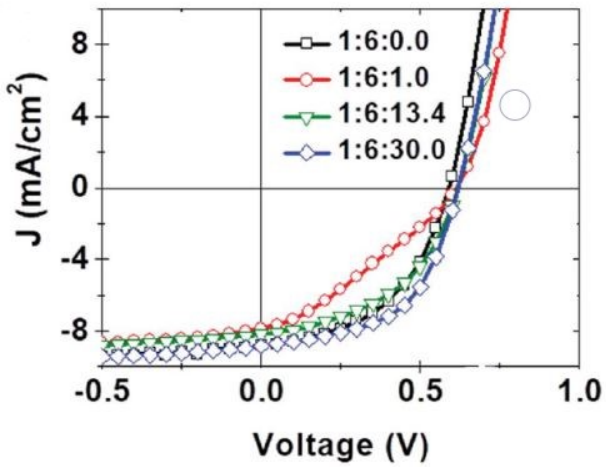

Figure 2: Current Density vs. Voltage

The two characteristics obtained experimentally in the parent study are shown in Figures 1 and 2.

The experiments for each characteristic were divided into two: one with a single input and a single output and the other one with two inputs (second one being the concentration of PFI) and one output.

Figure 3: Experiment Design

Thus, depending on whether the characteristic is T% vs. λ or J vs. V, the experiments were designed as shown in Figure 3.

For the single-input experiments, the ANN is trained and tested using data corresponding to any one PFI concentration. The best-architecture is found using that concentration and the final ANN is used on the data for other concentrations. For the two-input experiments, all available data points are used for training and testing to find the best architecture. To judge the reliability of the final ANN, a new PFI concentration is given to the ANN to predict its transmittance/current density. True output values for this concentration are not known and therefore it is not possible to use performance metrics such as mean absolute error (MAE) etc. to judge its performance numerically. However, the graph corresponding to the new concentration is

plotted on the existing figure obtained experimentally. The position of the new line with respect to other concentrations' lines gives a rough idea about the accuracy and reliability of the ANN.

The numbers of samples to ANN for single-input experiments are between 26 and 27, while for two-input experiments they are 100 for J vs. V and 162 for T% vs. λ respectively. In the case of FLANN, each feature was expanded into five features using a polynomial. The relatively low numbers of samples were split for training and testing into 70% and 30% of the total samples. This meant that the ANN was trained with as low as 19 samples. This limited the ANN from having a vast pool of examples to learn from.

3.2 Multilayer Perceptron

Multilayer perceptrons are neural networks with one or more hidden layers between the input layer and output layer. Each layer is composed of a number of nodes or neurons which are processing units for the data they receive from the previous layer. The input layer doesn't have a preceding layer and therefore accepts the data samples directly and has neurons equal to the number of features in the dataset. The neurons in the subsequent layers are connected to the previous layers' neurons with connecting links, each associated with a number called weight. The weight multiples the output of a neuron and gives the multiplied value as an input to the next neuron, which then modifies the input according to its activation function.

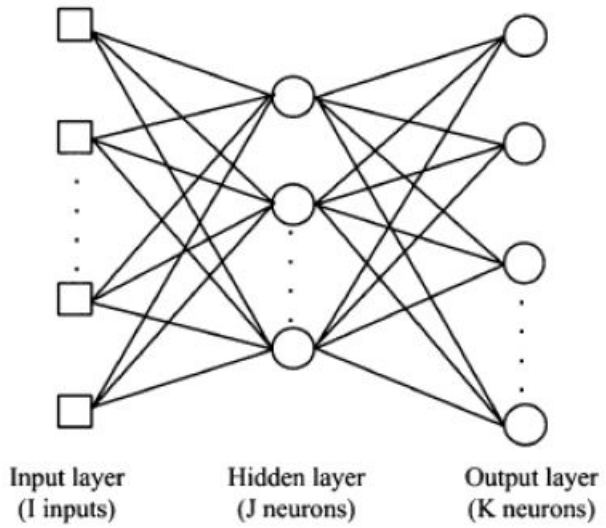

Figure 4: Multilayer Perceptron

Figure 4 shows the general architecture of an ANN. A fully connected neuron network has no broken links between any two layers. The more number of features and more complex the pattern to be learned, the

greater number of layers and neurons may be required. The neural network learns during the training phase where it accepts the training samples and outputs the predicted values using a set of randomly assigned weights. The predicted values are compared with the true values and the errors are used to slightly modify the weights. The network again outputs some values to be compared with true values, and to be used to again modify the weights in the direction of error reduction. This iterative process employs back propagation algorithm and stops when the network has converged i.e. the predicted and true values come sufficiently close numerically.

A number of training algorithms exist that can optimize the performance function. These optimizers are based on either the gradient of network performance or the Jacobian of errors. Modifications in these two classes of backpropagation algorithms result in a number of possible optimizers.

3.3 Functional Link Neural Network

Functional link neural networks are single-neuron neural networks that have an additional block preceding the neuron for the purpose of functionally expanding the initial features. The number of features being supplied to the neuron increases by a factor equal to the number of polynomials being used for functional expansion. FLANNs are based on the Cover theorem which states that projecting the features into a higher dimensional space increases the probability of linearly separating the training samples.

Therefore, if the input vector is:

$$X = [x_1, x_2, \ldots x_n] \tag{1}$$

Functional expansion will expand each individual variable of a sample into k variables. The final number of inputs to the neuron will then be nk

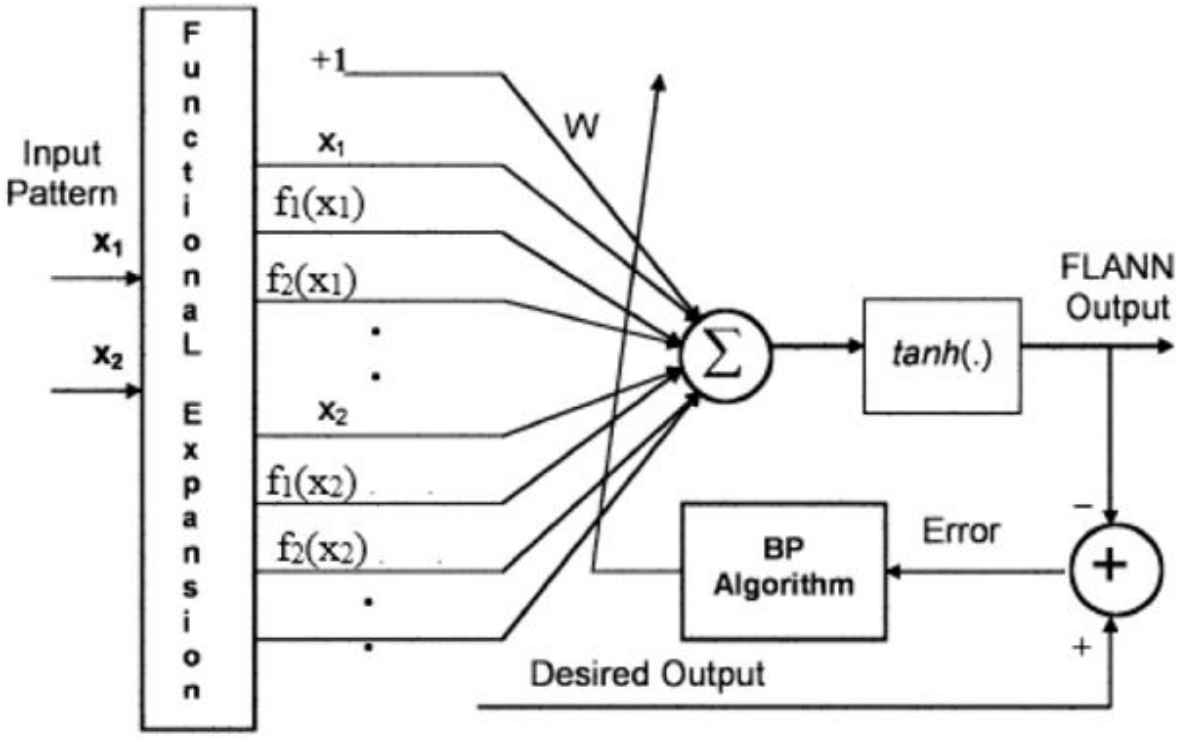

Figure 5: Functional Link Neural Network

Figure 5 shows each of the two inputs being expanded by the means of functions f_i where $i \in [0, 1, \ldots k-1]$. In the present study, the functions used are Chebyshev, Legendre, Power and Trigonometric as given in Equations 2, 3, 4 and 5 respectively

$$X_i = [1, x_i, (2x_i^2 - 1), (4x_i^3 - 3x_i), (8x_i^4 - 8x_i^2 + 1)] \tag{2}$$

$$X_i = [1, x_i, (3x_i^2 - 1)/2, (5x_i^3 - 3x_i)/2, (35x_i^4 - 30x_i^2 + 3)/8] \tag{3}$$

$$X_i = [1, x_i, x_i^2, x_i^3, x_i^4] \tag{4}$$

$$X_i = [x_i, \cos x_i, \sin x_i, \cos 2\pi x_i, \sin 2\pi x_i] \tag{5}$$

Table 1: Computational Complexity

Operator	**No. of Operations (MLP)**	**No. of Operations (FLANN)**
Addition	2HI + 3IK + 3K	2K(G+1) + K
Multiplication	3HI + 4IK + 3I + 5K	3K(G+1) + 2K
tanh(.)	I + K	K

The purpose of using FLANNs is to reduce training time and make the prediction more computationally efficient.

3.4 Complexity Comaprison of FLANN and MLP

The complexity of FLANN is lower than that of MLP. The greater the difference in network sizes of MLP and FLANN, the greater will be the reduction in complexity. The steps that each epoch will undergo in the training process are:

- Feedforward calculations to produce the output
- Error Calculations for each layer
- Modification of weights and biases during backpropagation

To mathematically compare the computational complexities of MLP and FLANN, consider a FLANN with architecture $G - K$ where G is the number of input variables to the neuron and K is the number of outputs (one in this study). For the same prediction problem, the corresponding MLP has an architecture $H - I - K$ where H is the reduced number of input variables prior to functional expansion, I is the number of neurons in the hidden layer and K is the same as described above i.e. the number of outputs.

The comparison has been shown in Table 1

Table 2: Performance of MLP with varying number of neurons

Nh	Overall MSE (dB)	Test MSE (dB)	Train MSE (dB)	Avg. R Squared	Avg. Train Time (seconds)
3	-15.87	-13.70	-17.24	0.980	1.81
4	-16.18	-13.74	-17.84	0.983	1.82
5	-16.15	-13.61	-17.90	0.983	1.81
6	-16.18	-13.61	-17.98	0.983	2.02
7	-16.05	-13.54	-17.79	0.983	2.02
8	**-15.77**	**-13.04**	**-17.76**	**0.983**	**1.88**
9	-15.82	-13.19	-17.69	0.982	2.27
10	-15.75	-13.05	-17.70	0.982	2.34
11	-15.72	-13.03	-17.65	0.982	2.36
12	-16.13	-13.32	-18.22	0.984	2.22
13	-16.15	-13.35	-18.23	0.984	1.92
14	-16.20	-13.40	-18.28	0.985	2.05
15	-16.16	-13.35	-18.26	0.985	1.93

4 Results and Analysis

4.1 Trasmittance Experiment 1.1

This experiment involved training the neural network with just the wavelengths and predicting the transmittance (%). The initial training was done on data corresponding to PFI concentration equal to 13.4. This training results were used to find the best architecture in terms if lowest Mean Squared Error (MSE), highest R squared and lowest training times. Since choosing an architecture by training the network just once isn't reliable, the training was done repeatedly 100 times. The MSEs, R squared values and the training times were then averaged over 100 ensembles. The results are recorded in Table 2.

As can be seen, 8 neurons is an appropriate choice, given that it gives appreciably low MSEs in a comparatively low training time. Using the $1 - 8 - 1$ architecture, its performance for concentration 13.4 in shown in Figure 6.

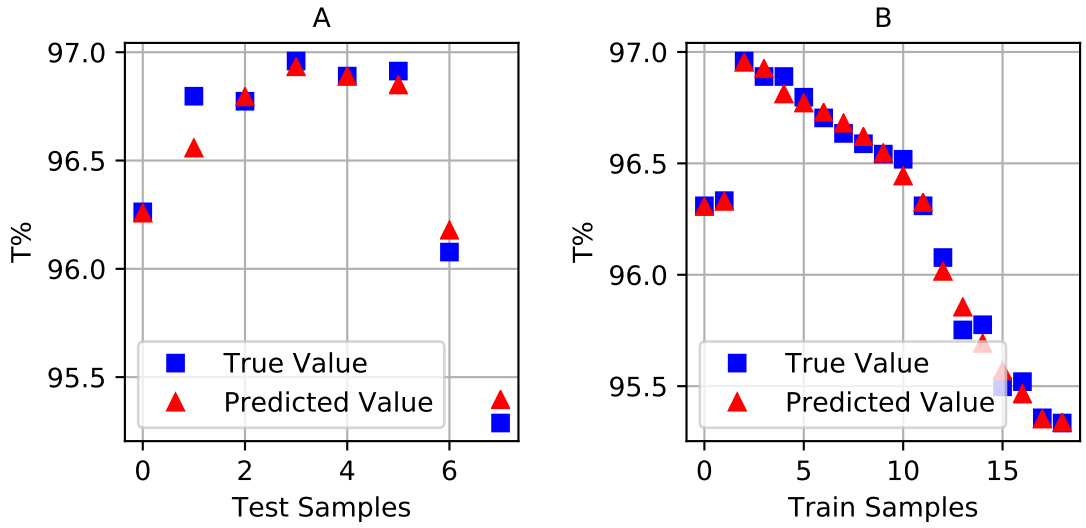

Figure 6: **MLP results**: MLP Perfromance for PFI concentration 13.4 (A) Testing Dataset, and (B) Training Dataset

The performances on other concentrations are plotted in Figures 7, 8, 9, 10 and 11 for PFI concentrations 0, 1, 2.5, 6 and 30 respectively. This gives an idea for the overall performance of the network for any given concentration.

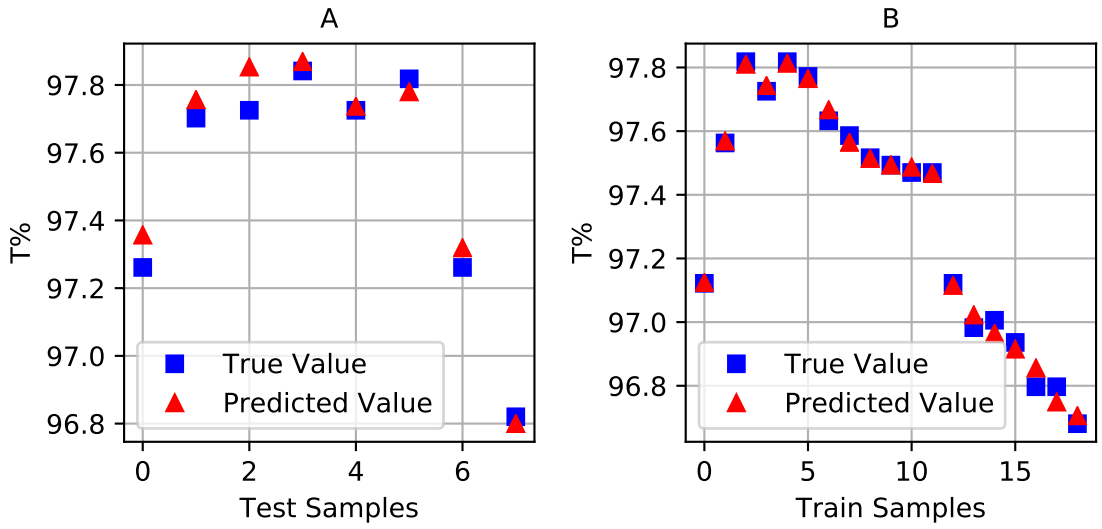

Figure 7: **MLP results**: MLP Perfromance for PFI concentration 0 (A) Testing Dataset, and (B) Training Dataset

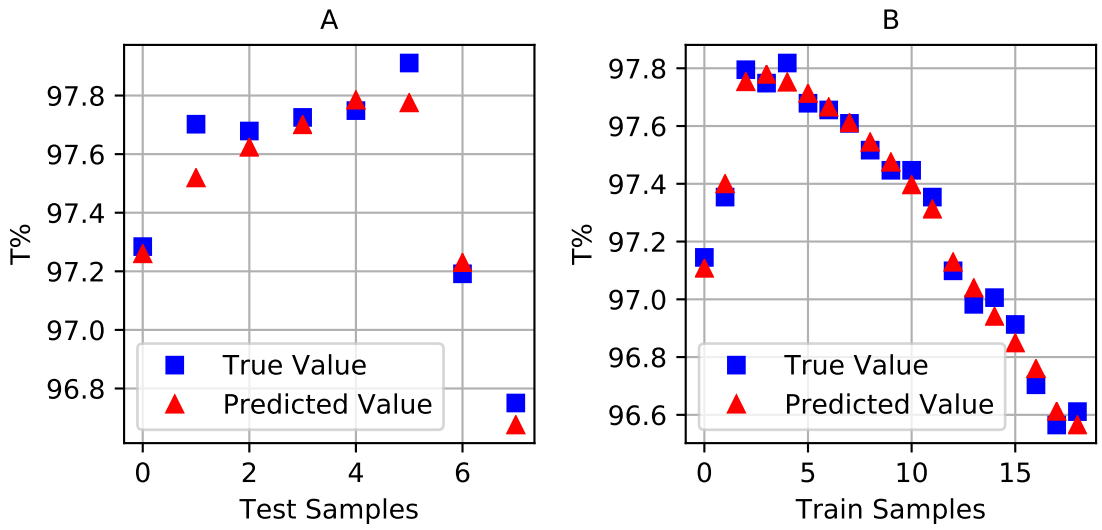

Figure 8: **MLP results**: MLP Perfromance for PFI concentration 1.0 (A) Testing Dataset, and (B) Training Dataset

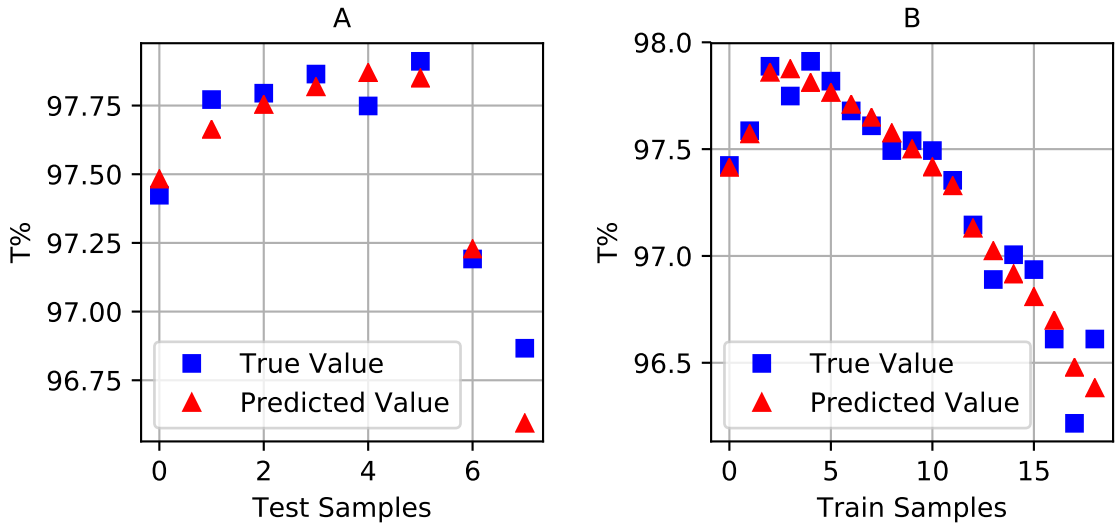

Figure 9: **MLP results**: MLP Perfromance for PFI concentration 2.5 (A) Testing Dataset, and (B) Training Dataset

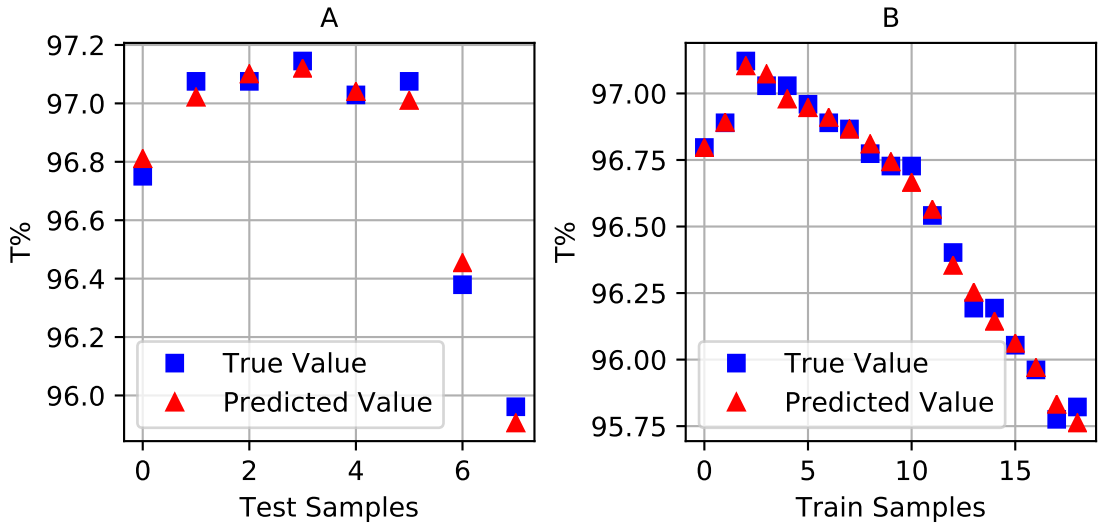

Figure 10: **MLP results**: MLP Perfromance for PFI concentration 6 (A) Testing Dataset, and (B) Training Dataset

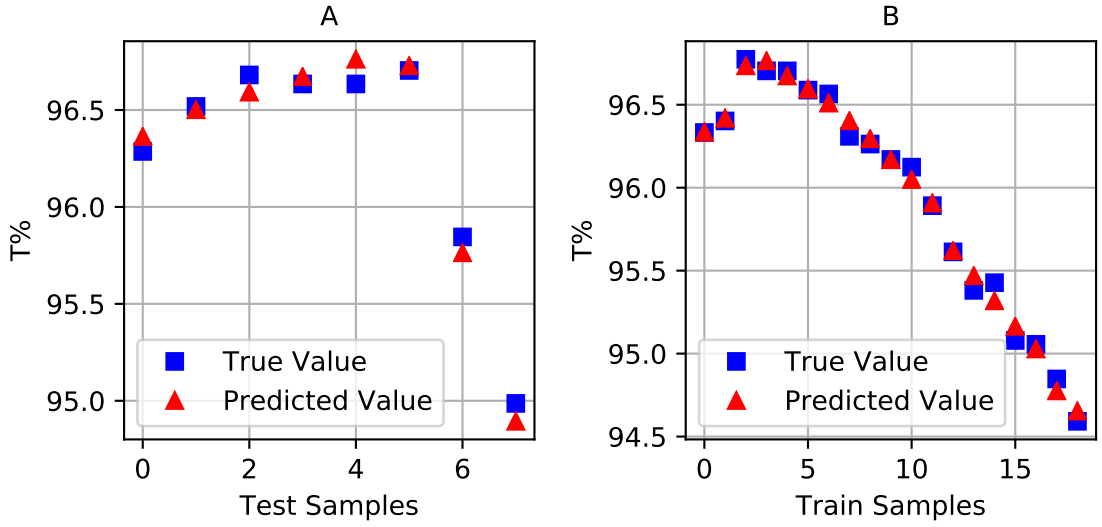

Figure 11: **MLP results**: MLP Perfromance for PFI concentration 30 (A) Testing Dataset, and (B) Training Dataset

4.2 Transmittance Experiment 1.2

The exact same experiment was repeated but this time using FLANN. The first step was to choose the better performing FLANN polynomial from among Chebyshev, Legendre, Power and Trigonometric, each having 5 polynomials. The bar charts given in Figure 12 show the performance of each FLANN polynomial in terms of average MSE and average training time over 100 ensembles. Chebyshev was chosen as it gave the most negative MSE in dB (lowest absolute value of MSE).

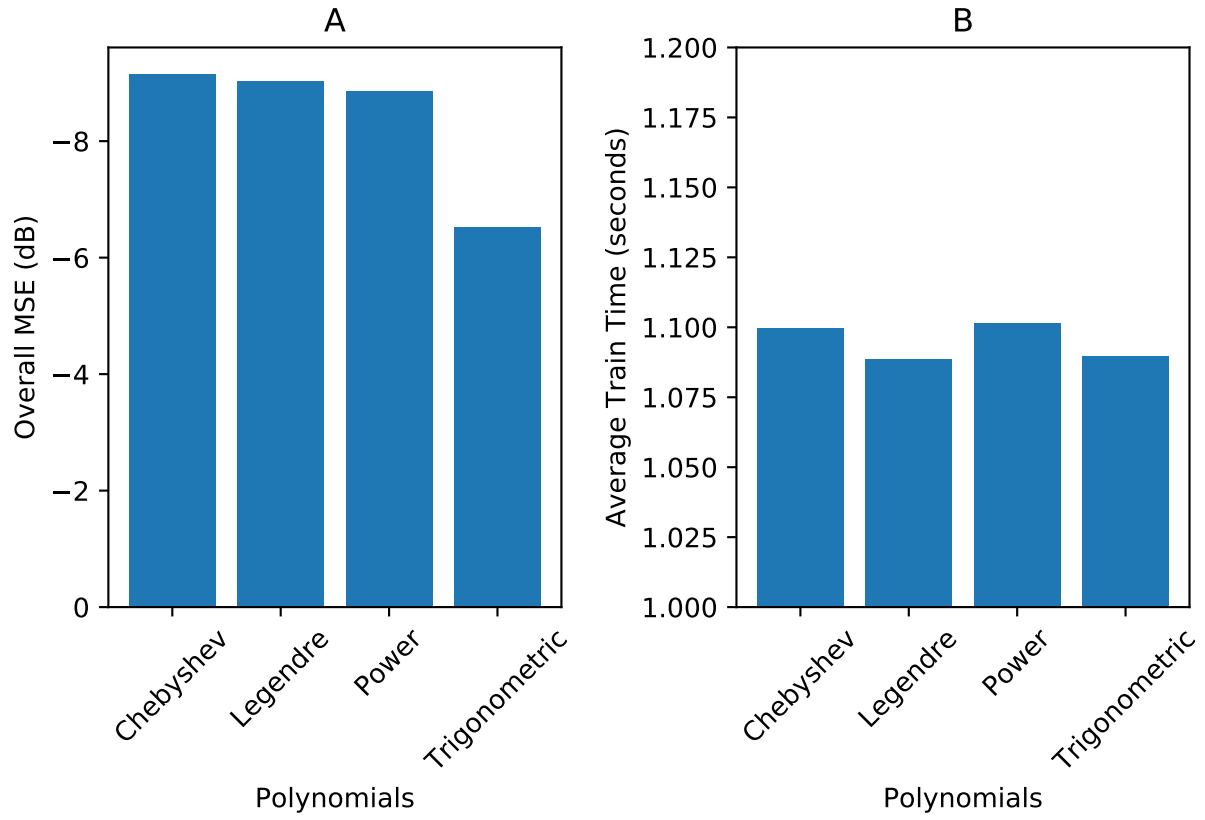

Figure 12: **Choosing the best FLANN Polynomial**: Comparison Using (A) Resulting Mean Squared Error (MSE) and, (B) Average training time.

As in the case of MLP, the FLANN performance on PFI concentration 13.4, which is the concentration used to make the bar charts, is shown in 13.

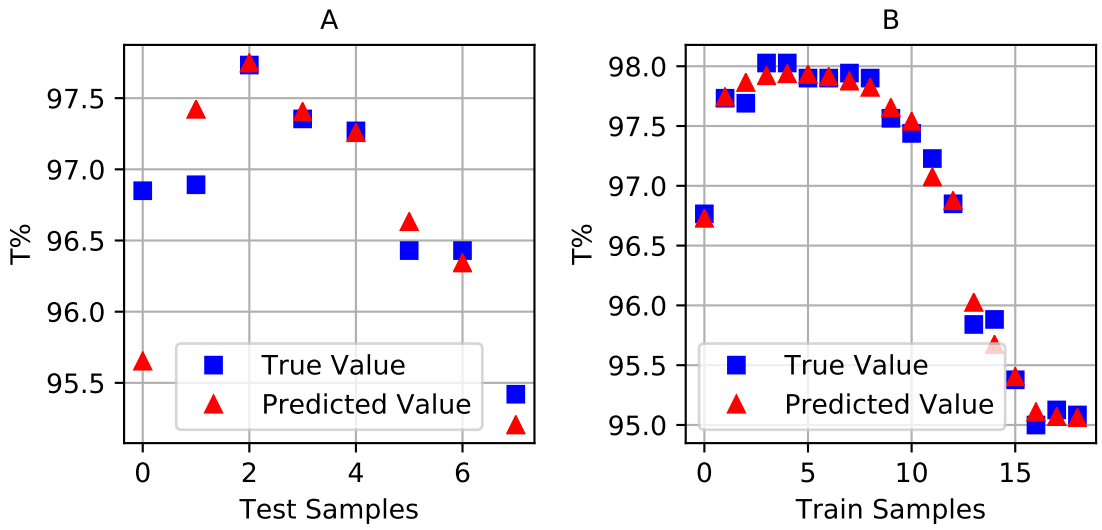

Figure 13: **FLANN results**: FLANN Perfromance for PFI concentration 13.4 (A) Testing Dataset, and (B) Training Dataset

The performance on other concentrations are shown in Figures 14, 15, 16, 17 and 18 for concentration 0, 1, 2.5, 6 and 30 respectively.

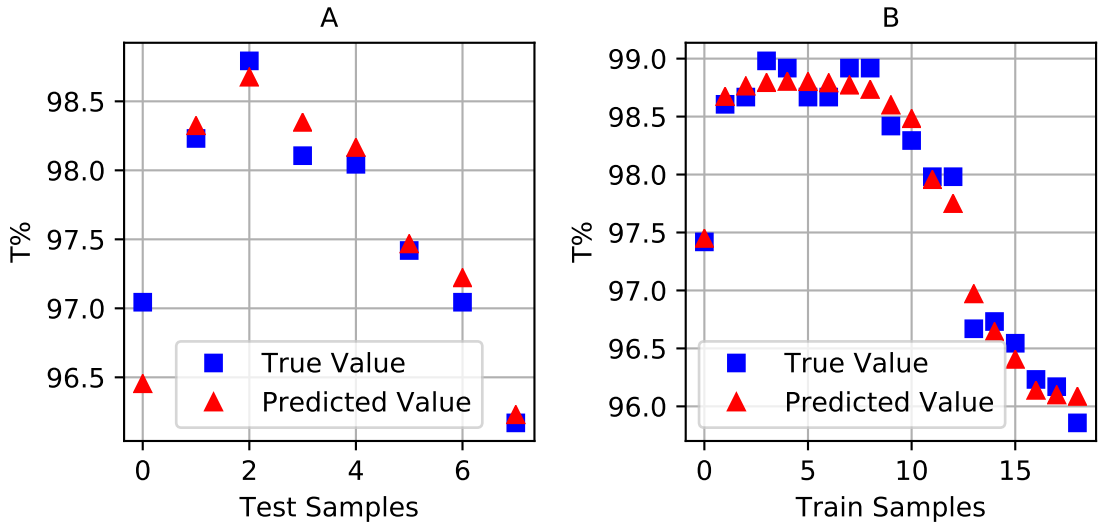

Figure 14: **FLANN results**: FLANN Perfromance for PFI concentration 0 (A) Testing Dataset, and (B) Training Dataset

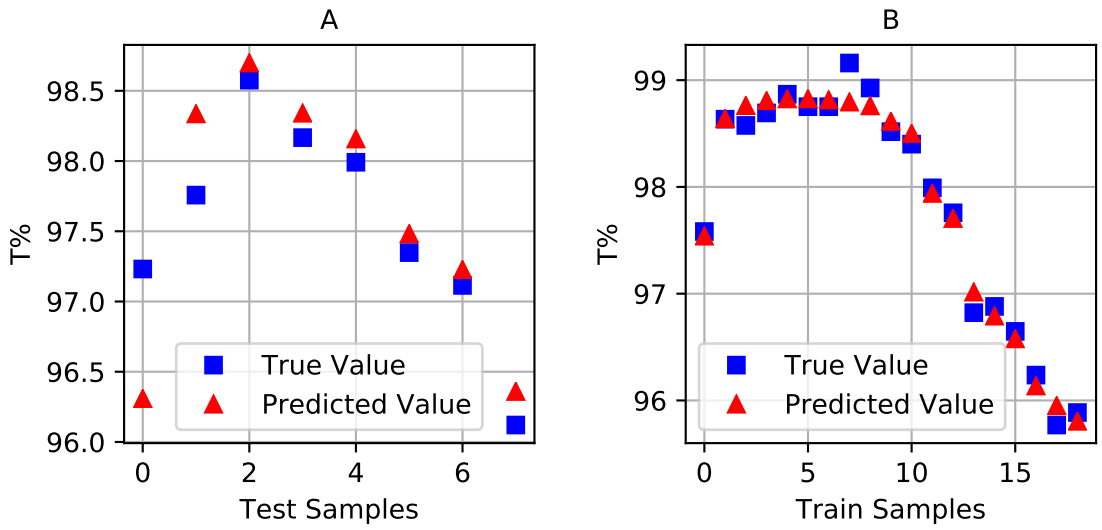

Figure 15: **FLANN results**: FLANN Perfromance for PFI concentration 1 (A) Testing Dataset, and (B) Training Dataset

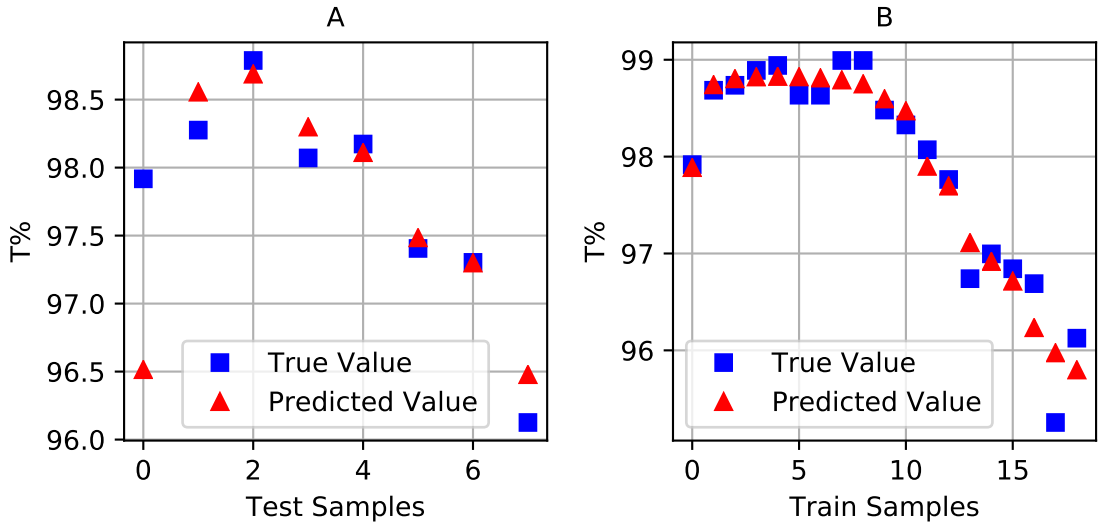

Figure 16: **FLANN results**: FLANN Perfromance for PFI concentration 2.5 (A) Testing Dataset, and (B) Training Dataset

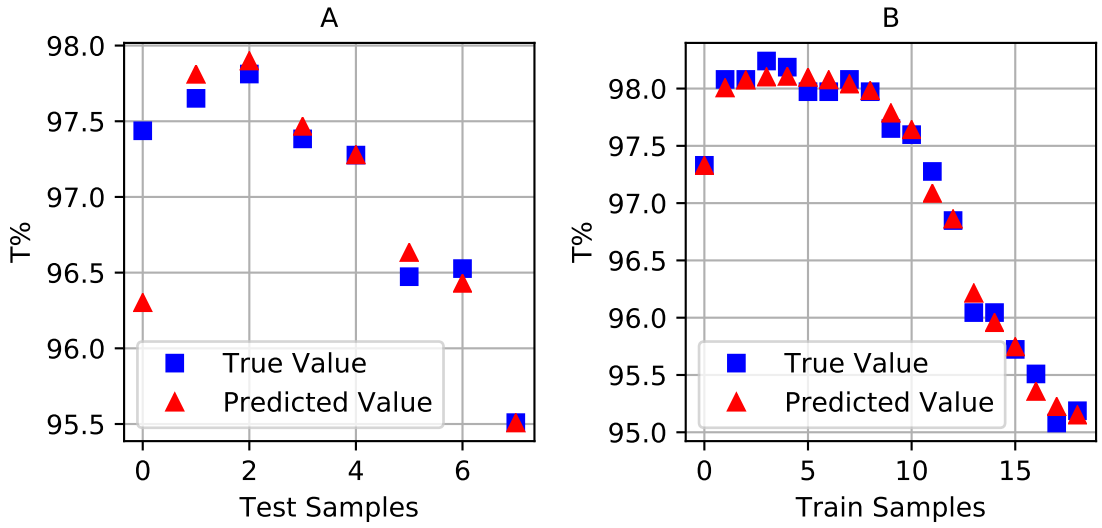

Figure 17: **FLANN results**: FLANN Perfromance for PFI concentration 6 (A) Testing Dataset, and (B) Training Dataset

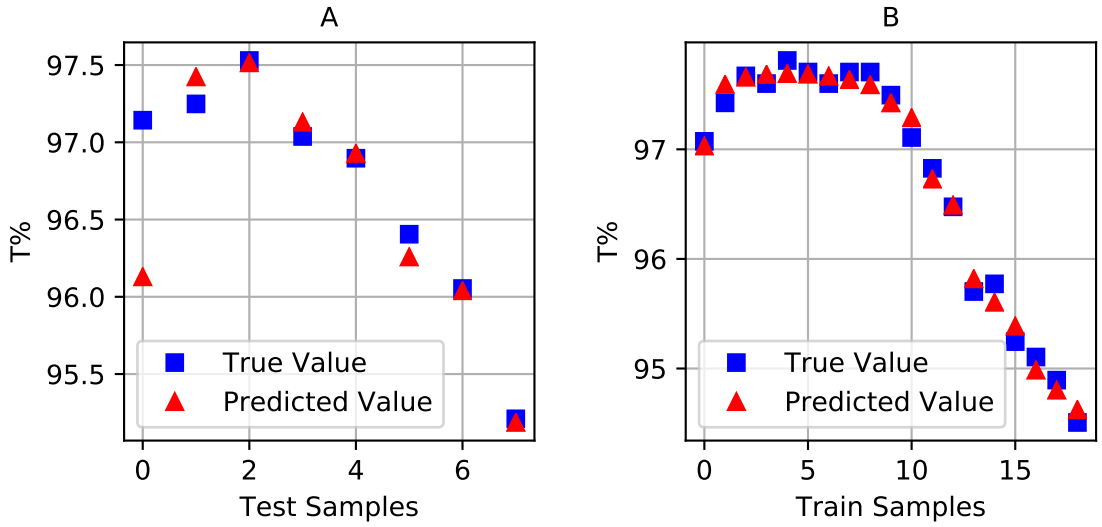

Figure 18: **FLANN results**: FLANN Perfromance for PFI concentration 30 (A) Testing Dataset, and (B) Training Dataset

Table 3: Performance of MLP with varying number of neurons

Nh	Overall MSE (dB)	Test MSE (dB)	Train MSE (dB)	Avg. R Squared	Avg. Train Time (seconds)
3	-13.50	-13.08	-13.70	0.957	1.17
4	-17.42	-16.73	-17.76	0.983	1.16
5	-17.56	-16.71	-17.98	0.984	1.17
6	-18.50	-17.33	-19.12	0.988	1.18
7	-18.62	-17.61	-19.13	0.988	1.23
8	-18.37	-17.14	-19.03	0.987	1.33
9	-18.95	-17.79	-19.57	0.989	1.34
10	-18.72	-17.52	-19.37	0.988	1.23
11	-18.95	-17.80	-19.57	0.989	1.23
12	**-19.07**	**-17.91**	**-19.69**	**0.989**	**1.27**
13	-18.94	-17.79	-19.54	0.989	1.33
14	-18.76	-17.69	-19.32	0.988	1.34
15	-18.82	-17.66	-19.43	0.988	1.35
16	-19.10	-17.91	-19.74	0.989	1.35
17	-19.04	-17.92	-19.63	0.989	1.37
18	-19.01	-17.82	-19.65	0.989	1.36
19	-18.79	-17.68	-19.38	0.988	1.30

4.3 Transmittance Experiment 1.3

The second part of the experiment with transmittance had two inputs to the ANN i.e. wavelength and the concentration. That means that the ANN will be trained to predict the Transmittance (%) of PEDOT:PSS layer given the wavelength and PFI concentration present. For this, different architectures were compared to arrive at the better performing one. The hidden layer was limited to one and the numbers of neurons were varied within it. The performance of each architecture is shown in Table 3. The architecture with 12 neurons was chosen as it gave the lowest MSE, with similar training times as other architectures.

Using the $\{2-12-1\}$ architecture, the performance of the MLP is shown in 19 for both the test and training samples. As obvious, the performance can be seen to be better on training samples compared to the testing samples as the MLP is familiar with the training dataset examples and has used them to train itself, while the testing dataset examples are new to the network.

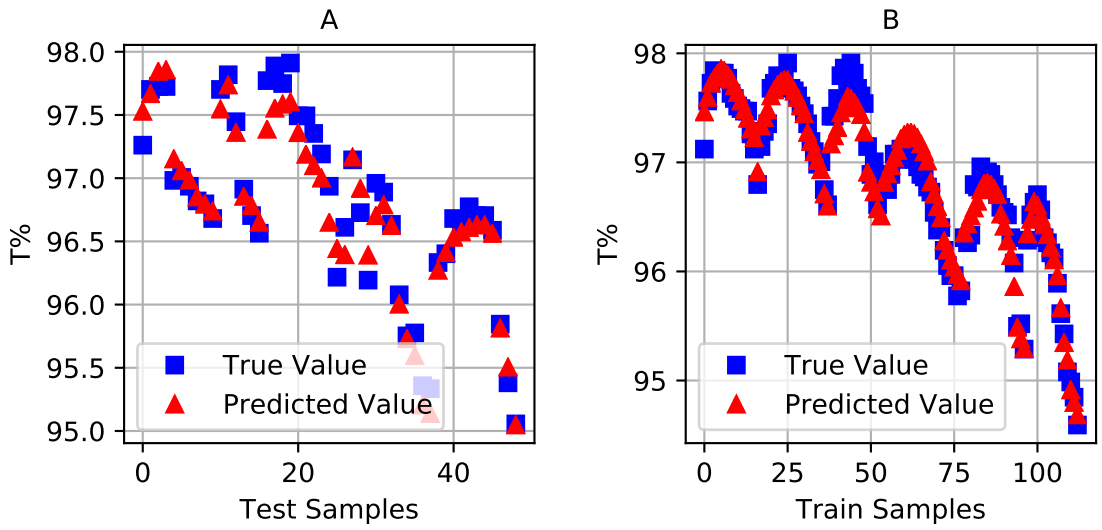

Figure 19: **MLP results**: MLP Perfromance on (A) Testing Dataset, and (B) Training Dataset

One possible application for such a network can be to predict the transmittance of the PEDOT:PSS layer with a specific PFI concentration prior to manufacturing the layer. The optical characteristics, if known beforehand, can assist in assessing the usefulness of the PEDOT:PSS layer in the practical applications at hand. With this in mind, the MLP was provided a new unknown concentration and wavelength values in the range given in Figure 1. The two PFI concentrations of 29 and 15 were chosen because of their numerical closeness to the already known concentrations of 30 and 13.4 respectively. This allowed the plotting of transmittance vs. wavelength characteristics of both concentrations, one new (i.e. 29 or 15) and one already known (30 or 13.4) on the same figure. Ideally, their lines should follow close to each other because both concentrations are almost equal. The results are given in Figure 20. As can be seen, the predictions seem pretty accurate as the two lines are very close to each other.

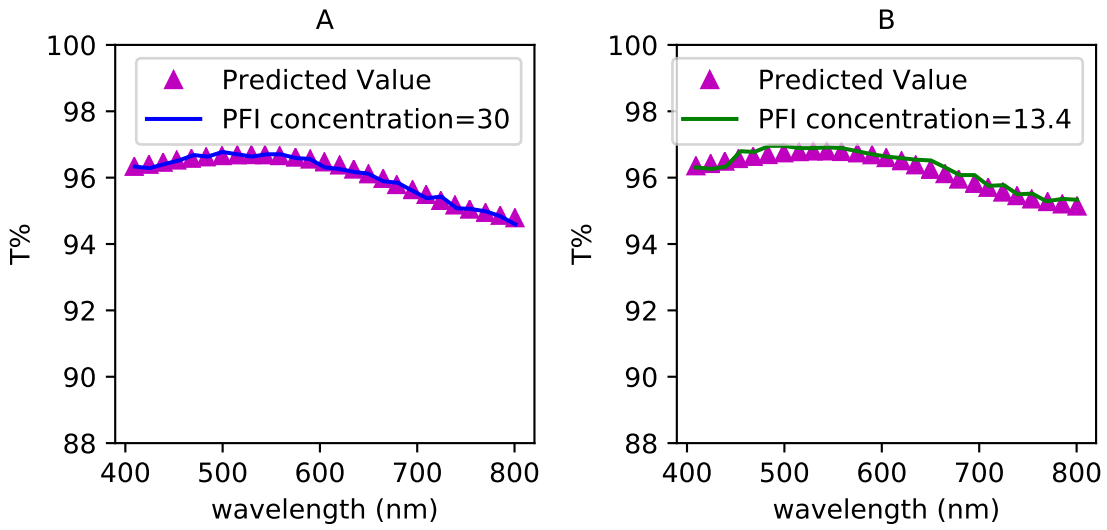

Figure 20: **Predicting Transmittance of a new PFI concentration** : (A) A new PFI concentration of 29, (B) A new PFI concentration of 15.

4.4 Transmittance Experiment 1.4

Based on the bar chart of Figure 21, Trigonometric FLANN has the lowest MSE. Therefore, we have chosen Trigonometric FLANN. Its performance on the dataset can be seen in Figure 22.

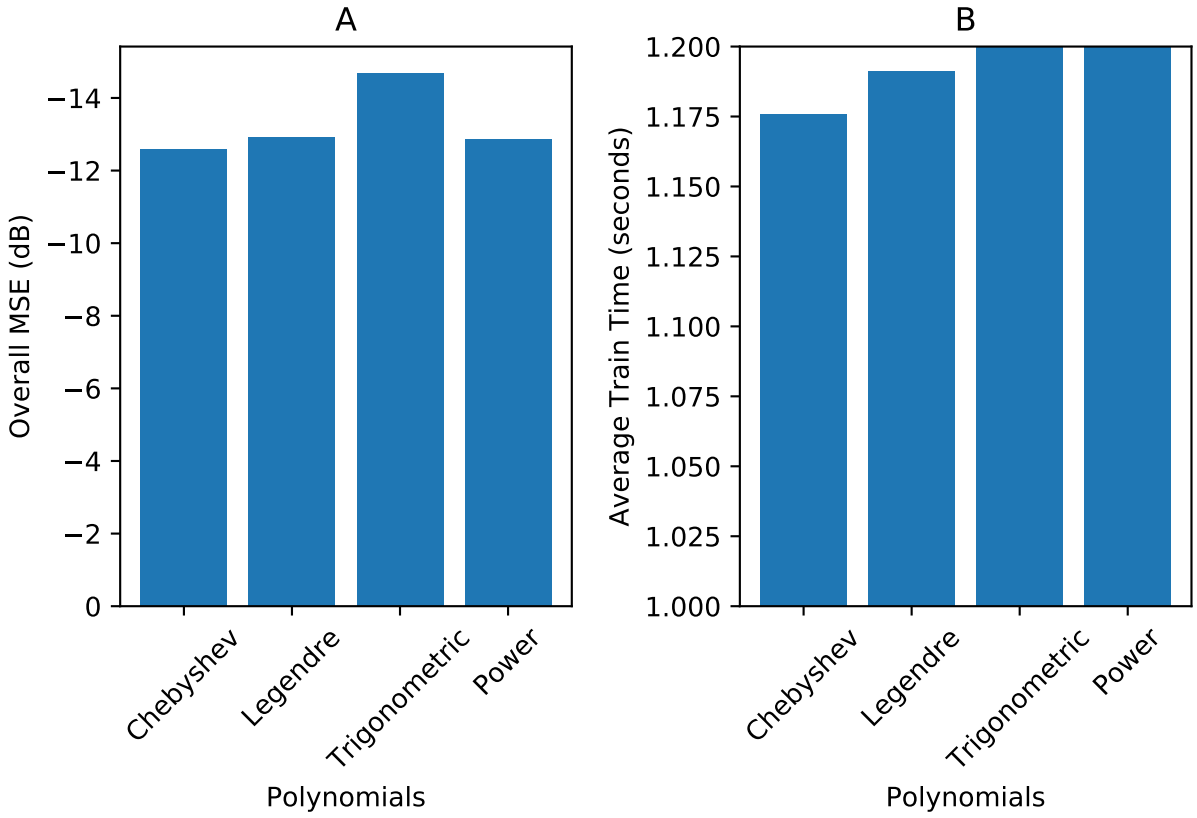

Figure 21: **Choosing the best FLANN Polynomial**: Comparison Using (A) Resulting Mean Squared Error (MSE) and, (B) Average training time.

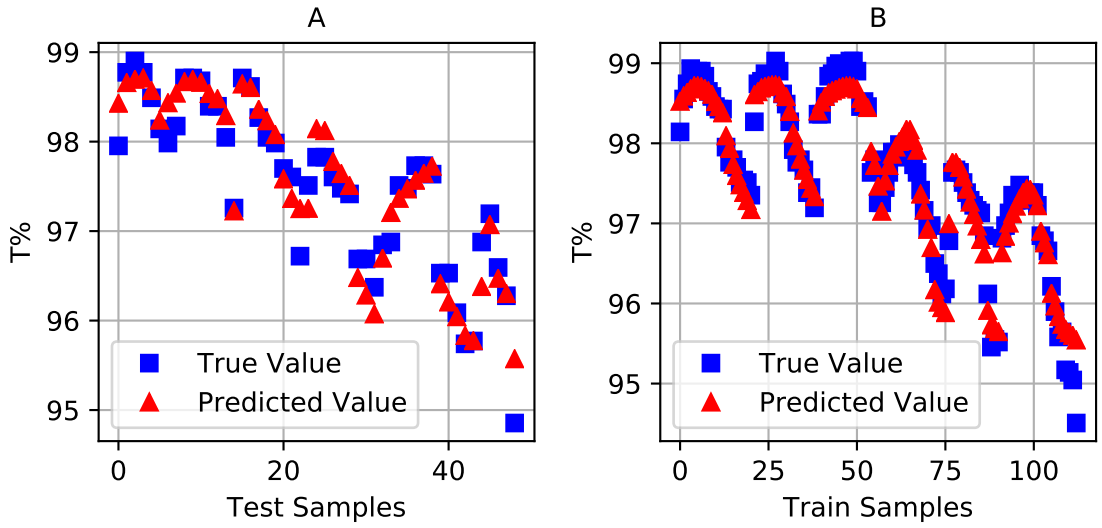

Figure 22: **FLANN results**: FLANN Perfromance for (A) Testing Dataset, and (B) Training Dataset

The transmittance percentage of new PFI concentrations 29 and 15 are shown in23. The performance of FLANN isn't as good as it was of the MLP. Notice how the predicted values fluctuate above and below the already known transmittance values of almost equal concentrations. For this purpose, MLP will be chosen as it gave a better, more reliable performance.

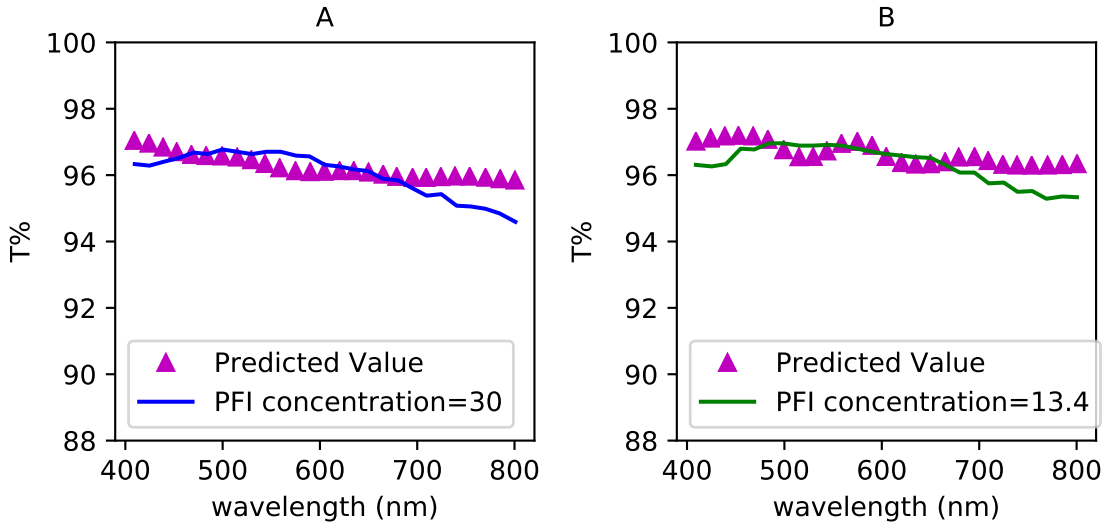

Figure 23: **Predicting Transmittance of a new PFI concentration** : (A) A new PFI concentration of 29, (B) A new PFI concentration of 15.

Table 4: Performance of MLP with varying number of neurons

Nh	Overall MSE (dB)	Test MSE (dB)	Train MSE (dB)	Avg. R Squared	Avg. Train Time
3	-22.27	-18.96	-24.82	0.997	1.05
4	-22.54	-19.26	-25.04	0.997	1.01
5	-22.96	-19.45	-25.84	0.997	1.04
6	-22.85	-19.35	-25.72	0.997	1.13
7	-23.04	-19.50	-25.96	0.997	1.14
8	**-23.02**	**-19.46**	**-25.98**	**0.997**	**1.13**
9	-23.26	-19.58	-26.43	0.998	1.13
10	-23.13	-19.50	-26.21	0.998	1.23
11	-23.02	-19.43	-26.04	0.997	1.14
12	-23.05	-19.51	-25.98	0.997	1.16
13	-22.91	-19.34	-25.90	0.997	1.17
14	-22.95	-19.38	-25.93	0.997	1.16
15	-22.95	-19.41	-25.88	0.997	1.18
16	-23.00	-19.43	-25.96	0.997	1.19
17	-22.99	-19.44	-25.93	0.997	1.12
18	-23.00	-19.46	-25.92	0.997	1.09
19	-23.20	-19.59	-26.24	0.997	1.10

4.5 Current Density Experiment 2.1

Experiments similar to the transmittance vs. wavelength characteristic were conducted for the current density vs. voltage characteristic as well. That means the first experiment involved predicting, for any one PFI concentration, the current density when voltage is provided as an input to the ANN. Just as before, the PFI concentration chosen was 13.4 in order to select the best performing architecture. The numbers of neurons in the hidden layer were varied and their effect on averages of MSE, R squared and training times over 100 ensembles were recorded in Table 4. The architecture $1 - 8 - 1$ gave high enough average MSEs for overall, training and test dataset with appreciable training times.

The actual vs. predicted values for concentration 13.4 are shown in 24.

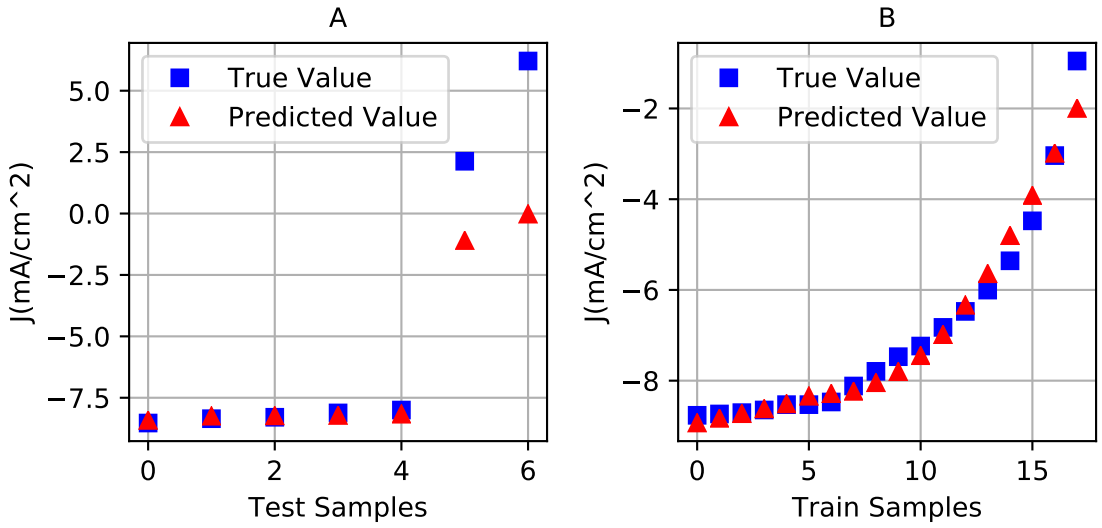

Figure 24: **MLP results**: MLP Perfromance for PFI concentration 13.4 (A) Testing Dataset, and (B) Training Dataset

The performance of the chosen MLP for PFI concentrations 0, 1 and 30 are shown in 25, 26 and 27 respectively.

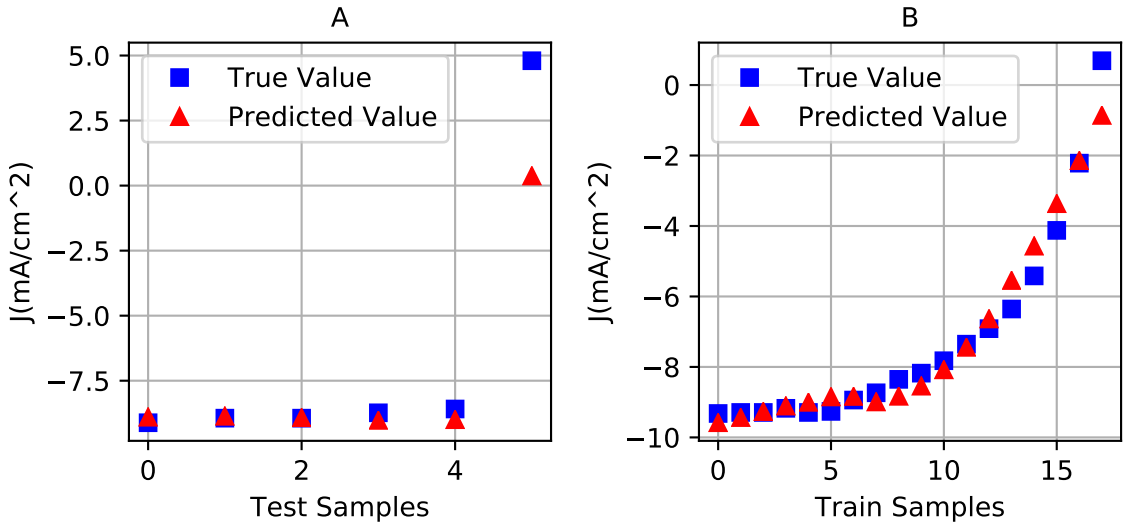

Figure 25: **MLP results**: MLP Perfromance for PFI concentration 0 (A) Testing Dataset, and (B) Training Dataset

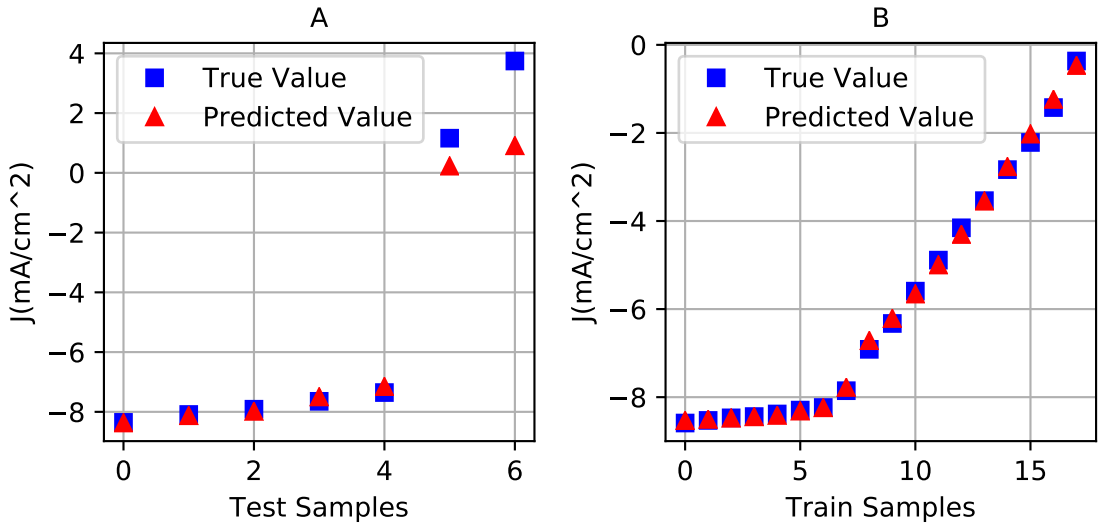

Figure 26: **MLP results**: MLP Perfromance for PFI concentration 1 (A) Testing Dataset, and (B) Training Dataset

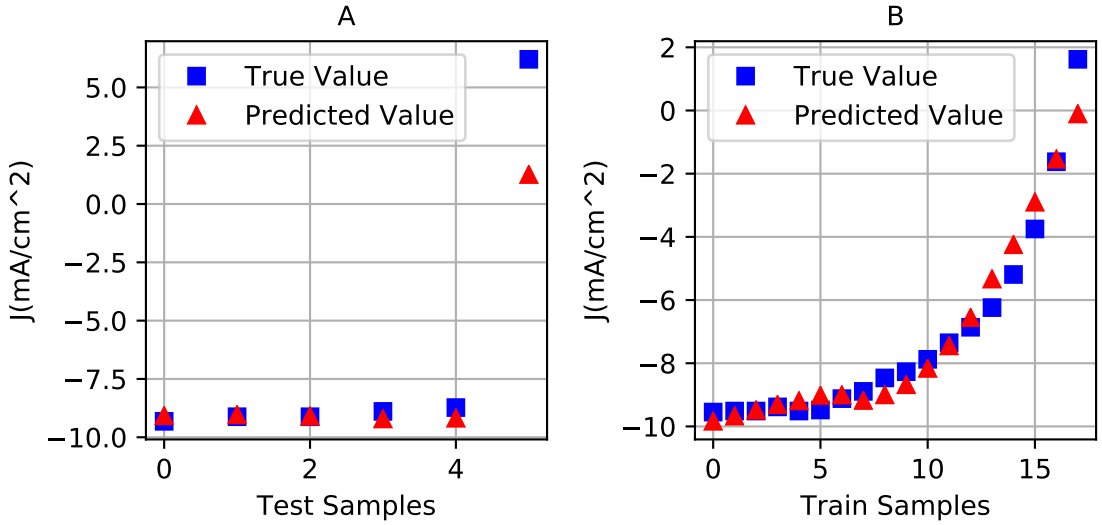

Figure 27: **MLP results**: MLP Perfromance for PFI concentration 30 (A) Testing Dataset, and (B) Training Dataset

4.6 Current Density Experiment 2.2

For the same problem, a FLANN was used instead of MLP. The different polynomials were tested for their suitability and the results are shown in Figure 28. As in the case of transmittance for the corresponding experiment, the preferred polynomial is Chebyshev due to its most negative average MSE and lowest average training time.

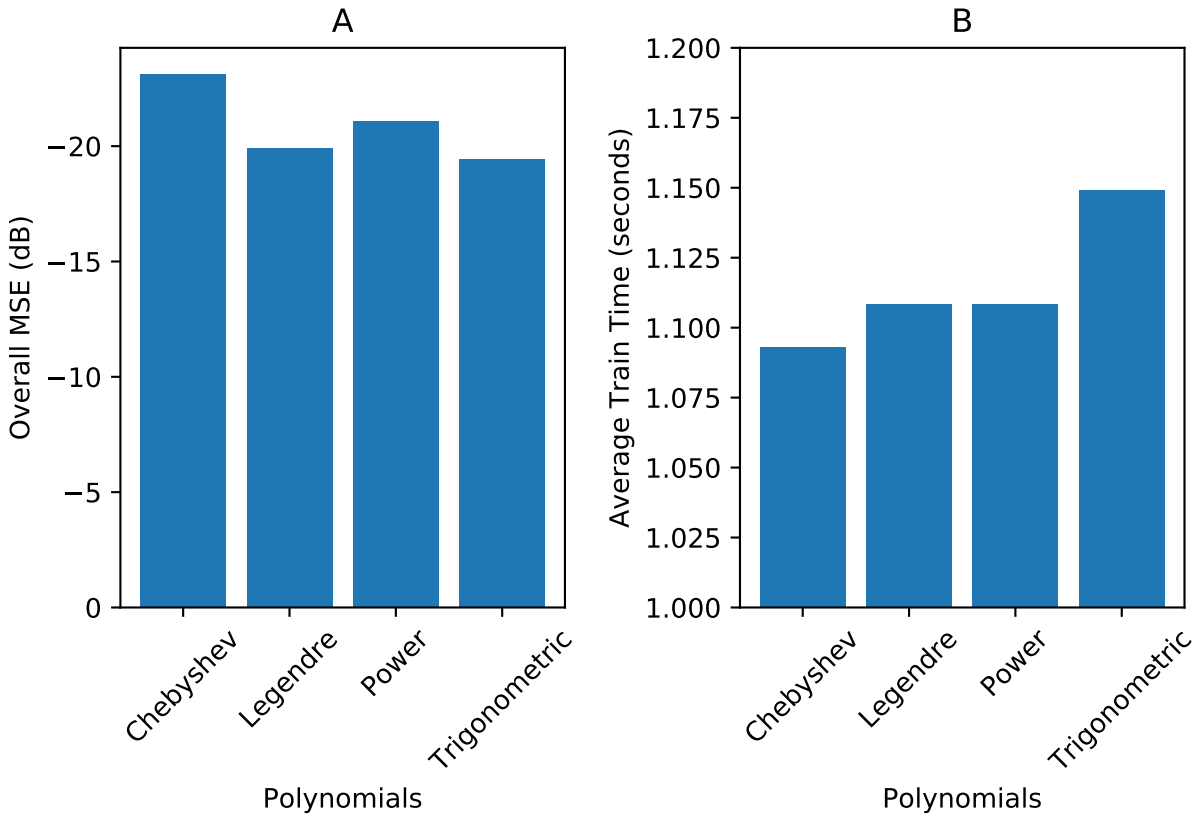

Figure 28: **Choosing the best FLANN Polynomial**: Comparison Using (A) Resulting Mean Squared Error (MSE) and, (B) Average training time.

Its performance for PFI concentration 13.4 is shown in Figure 29.

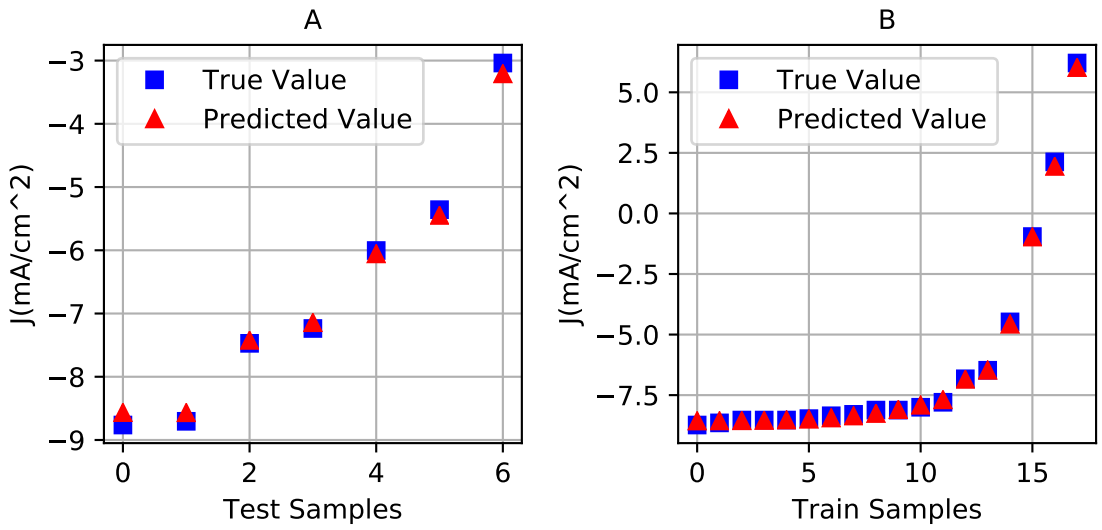

Figure 29: **FLANN results**: FLANN Perfromance for PFI concentration 13.4 (A) Testing Dataset, and (B) Training Dataset

Performance of Chebyshev FLANN on PFI concentrations 0, 1 and 30 are shown in Figures 30, 31 and 32 respectively.

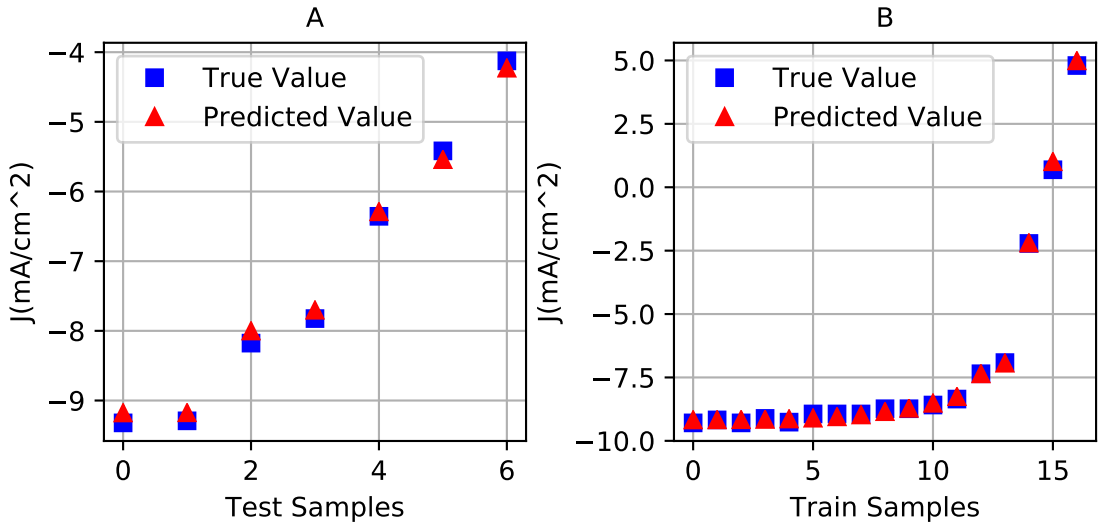

Figure 30: **FLANN results**: FLANN Perfromance for PFI concentration 0 (A) Testing Dataset, and (B) Training Dataset

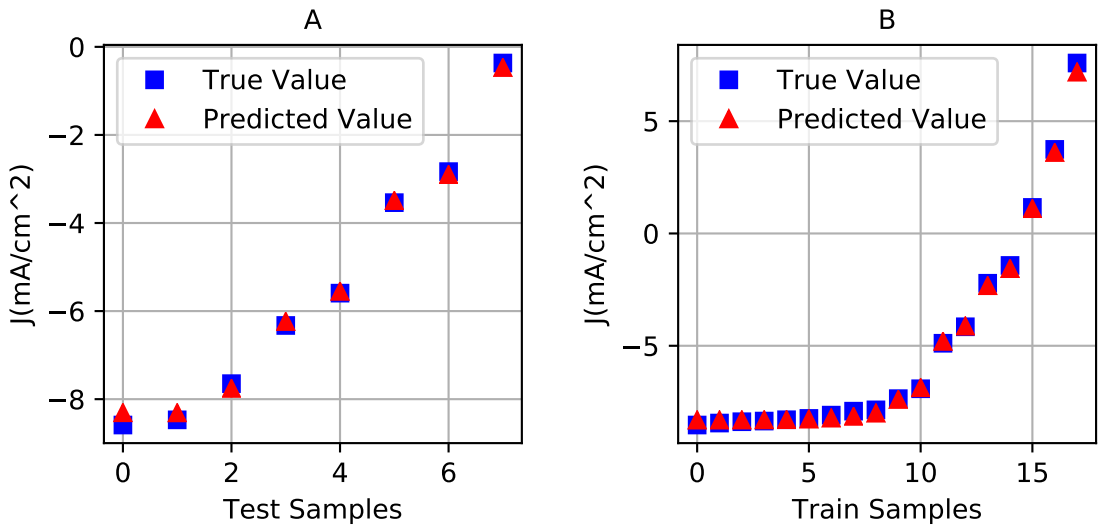

Figure 31: **FLANN results**: FLANN Perfromance for PFI concentration 1 (A) Testing Dataset, and (B) Training Dataset

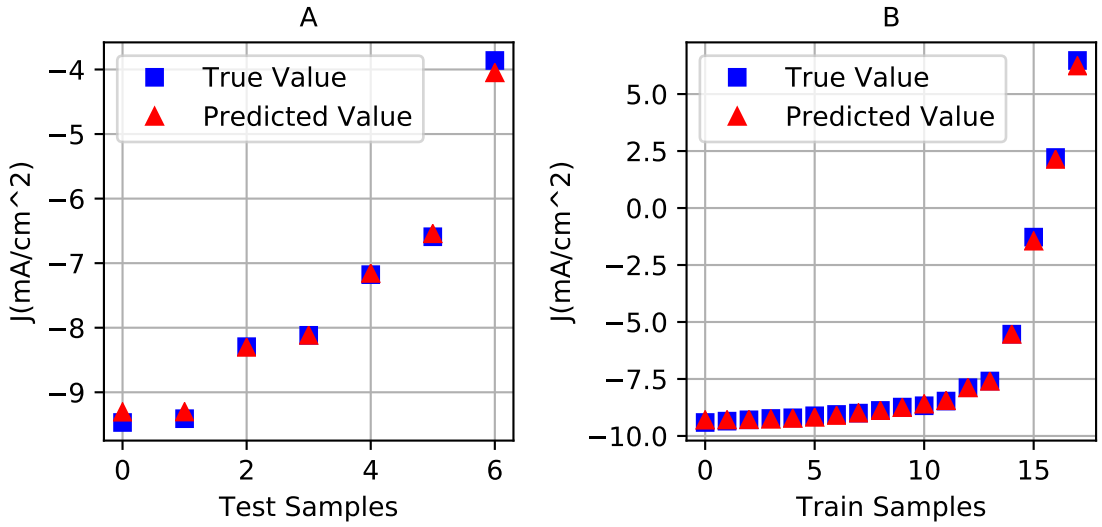

Figure 32: **FLANN results**: FLANN Perfromance for PFI concentration 30 (A) Testing Dataset, and (B) Training Dataset

Table 5: Performance of MLP with varying number of neurons

Nh	Overall MSE (dB)	Test MSE (dB)	Train MSE (dB)	Avg. R Squared	Avg. Train Time
3	-14.76	-13.84	-15.22	0.969	1.31
4	-15.11	-14.17	-15.59	0.972	1.31
5	-16.08	-14.42	-17.04	0.980	1.36
6	-16.94	-15.46	-17.77	0.983	1.31
7	-16.87	-15.49	-17.64	0.982	1.33
8	-19.67	-18.36	-20.38	0.991	1.32
9	-16.72	-15.41	-17.43	0.981	1.34
10	-16.57	-15.36	-17.20	0.980	1.34
11	-17.79	-16.47	-18.51	0.985	1.34
12	**-19.31**	**-17.93**	**-20.07**	**0.990**	**1.36**
13	-17.35	-15.82	-18.21	0.984	1.35
14	-16.84	-15.40	-17.64	0.982	1.36
15	-16.67	-15.30	-17.42	0.981	1.36
16	-19.41	-18.06	-20.14	0.990	1.37
17	-16.84	-15.62	-17.49	0.981	1.40
18	-16.94	-15.58	-17.68	0.982	1.38
19	-19.18	-17.76	-19.97	0.990	1.40

4.7 Current Density Experiment 2.3

The second experiment on current density vs. voltage involved predicting current density based on voltage and PFI concentration present in the HEL layer. The ANN therefore had two input neurons and the neurons in the hidden layer were varied as given in Table 5. The architecture $2 - 12 - 1$ was chosen due to its low average MSEs and high average R squared values.

The performance of the MLP is shown in Figure 33

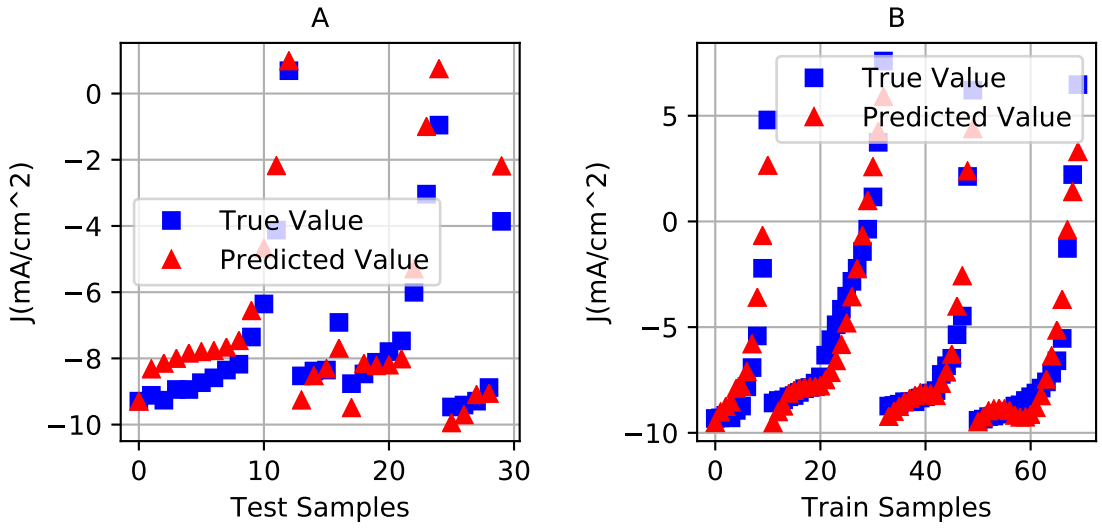

Figure 33: **MLP results**: MLP Perfromance on (A) Testing Dataset, and (B) Training Dataset

The MLP will be used for prediction for a new concentration for which we don't have the experimental data. The new PFI concentrations are 29 and 15 and the predicted values are shown against another numerically close, known concentration in Figure 34

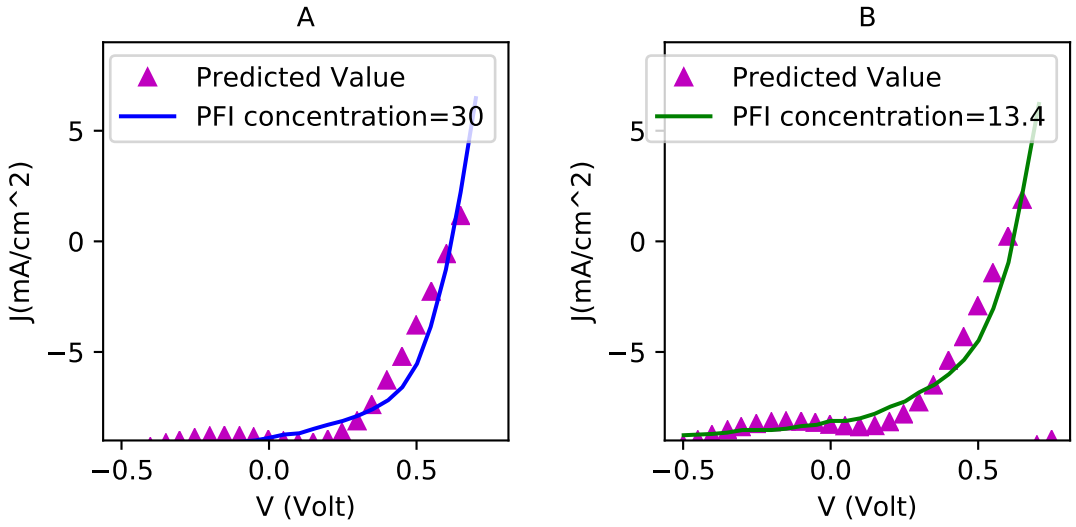

Figure 34: **Predicting Transmittance of a new PFI concentration** : (A) A new PFI concentration of 29, (B) A new PFI concentration of 15.

4.8 Current Density Experiment 2.4

As can be seen in 35, the Power polynomial gives good results in terms of both average MSE and average training time. Its performance is shown in Figure 36 for both test and training data.

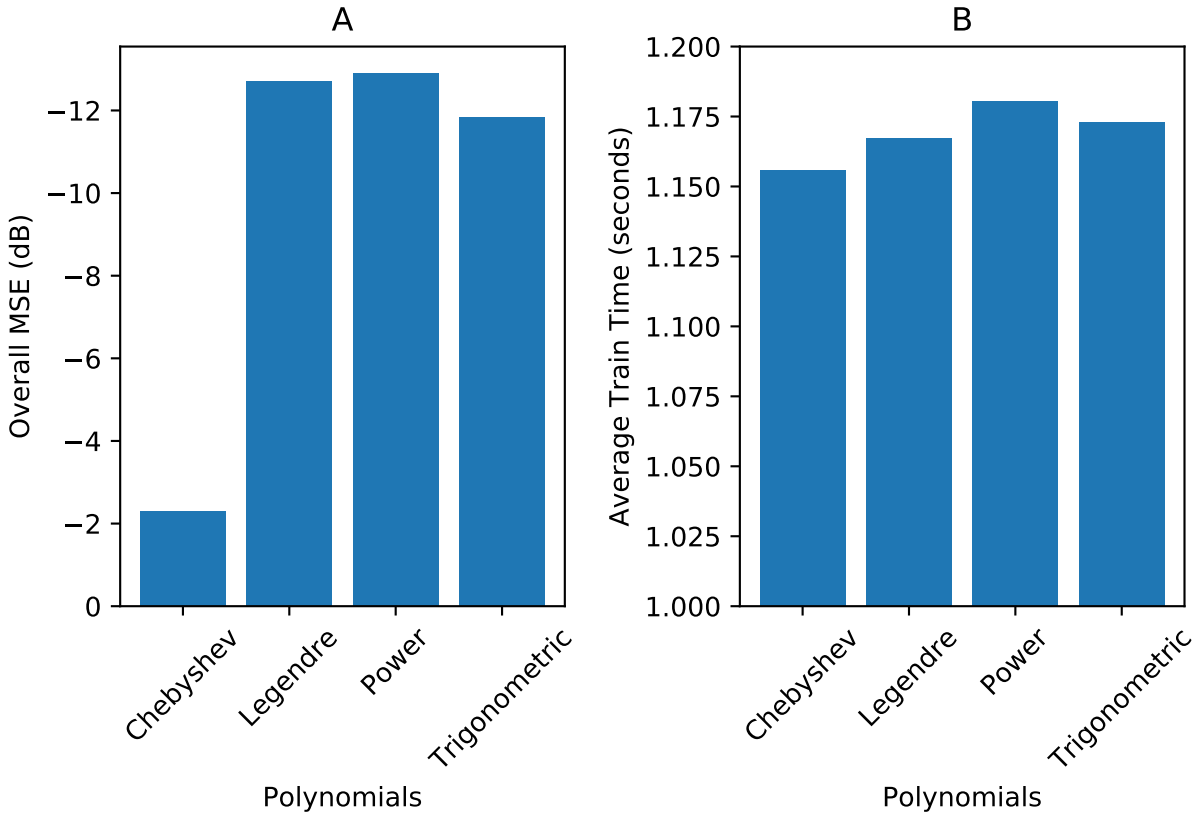

Figure 35: **Choosing the best FLANN Polynomial**: Comparison Using (A) Resulting Mean Squared Error (MSE) and, (B) Average training time.

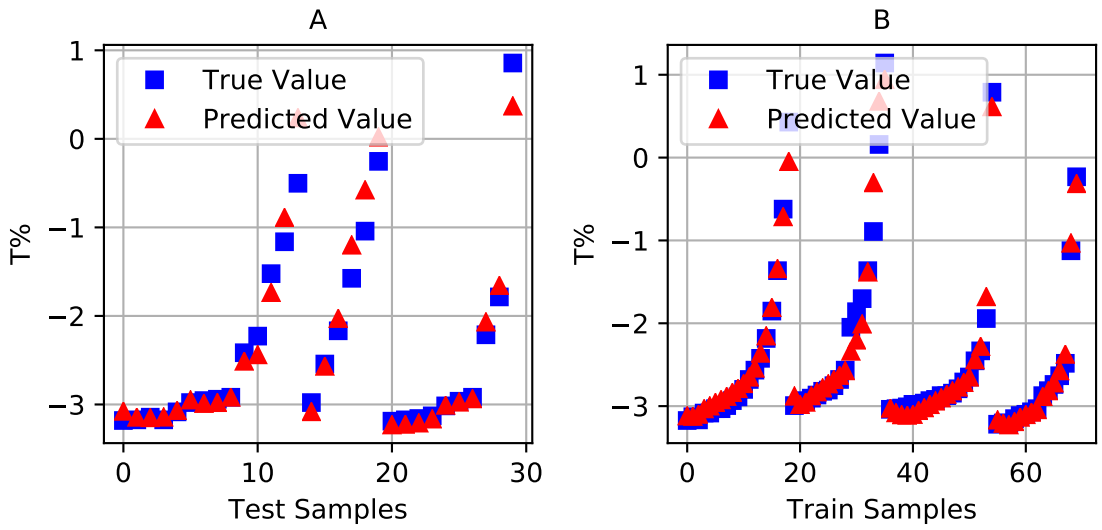

Figure 36: **FLANN results**: FLANN Perfromance for (A) Testing Dataset, and (B) Training Dataset

As done previously, the new PFI concentrations chosen are 29 and 15 and their results are given in 37. The new concentrations follow closely with the known concentrations , thus indicating that FLANN is working properly.

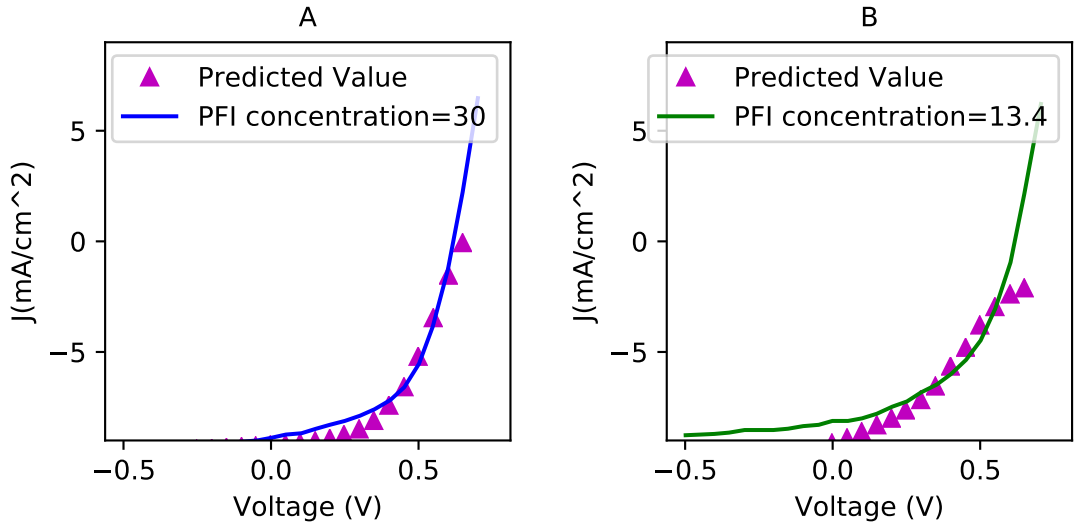

Figure 37: **Predicting Current Density of a new PFI concentration** : (A) A new PFI concentration of 29, (B) A new PFI concentration of 15.

5 Conclusion

The modeling of transmittance vs. wavelength and current density vs. voltage for OPVs containing PFI additive in the PEDOT:PSS hole extracting layer can be done to a reasonable accuracy. Except for the single input ANN in the case of current density vs. voltage characteristic, the accuracies obtained through MLP were always slightly higher than through FLANN. FLANN, however, had an edge in terms of having lower training times.

6 References

[1] C. N. Hoth, P. Schilinsky, S. A. Choulis, and C. J. Brabec, "Printing Highly Efficient Organic Solar Cells," *Nano Letters*, vol. 8, no. 9, pp. 2806–2813, Sep. 2008.

[2] M. Manceau, D. Angmo, M. Jørgensen, and F. C. Krebs, "ITO-free flexible polymer solar cells: From small model devices to roll-to-roll processed large modules," *Organic Electronics*, vol. 12, no. 4, pp. 566–574, Apr. 2011.

[3] F. C. Krebs, "Fabrication and processing of polymer solar cells: A review of printing and coating techniques," *Solar Energy Materials and Solar Cells*, vol. 93, no. 4, pp. 394–412, Apr. 2009.

[4] J. Ribierre, T. Aoyama, T. Muto, and P. André, "Hybrid organic–inorganic liquid bistable memory devices," *Organic Electronics*, vol. 12, no. 11, pp. 1800–1805, Nov. 2011.

[5] R. H. Kim *et al.*, "Non-volatile organic memory with sub-millimetre bending radius," *Nature Communications*, vol. 5, no. 1, p. 3583, May 2014.

[6] T. R. Andersen *et al.*, "Scalable, ambient atmosphere roll-to-roll manufacture of encapsulated large area, flexible organic tandem solar cell modules," *Energy & Environmental Science*, vol. 7, no. 9, p. 2925, Jun. 2014.

[7] K. A. Mazzio and C. K. Luscombe, "Correction: The future of organic photovoltaics," *Chemical Society Reviews*, vol. 44, no. 15, pp. 5744–5744, Jul. 2015.

[8] J. C. Ribierre *et al.*, "Ambipolar organic field-effect transistors based on solution-processed single crystal microwires of a quinoidal oligothiophene derivative," *Chem. Commun.*, vol. 51, no. 27, pp. 5836–5839, Mar. 2015.

[9] X. Gu *et al.*, "Roll-to-Roll Printed Large-Area All-Polymer Solar Cells with 5% Efficiency Based on a Low Crystallinity Conjugated Polymer Blend," *Advanced Energy Materials*, vol. 7, no. 14, p. 1602742, Jul. 2017.

[10] J. R. Y. Stevenson, S. Lattante, P. André, M. Anni, and G. A. Turnbull, "Fabrication of free-standing ordered fluorescent polymer nanofibres by electrospinning," *Applied Physics Letters*, vol. 106, no. 17, p. 173301, Apr. 2015.

[11] D. Joly, J.-W. Jung, I.-D. Kim, and R. Demadrille, "Electrospun materials for solar energy conversion: innovations and trends," *Journal of Materials Chemistry C*, vol. 4, no. 43, pp. 10173–10197, Nov. 2016.

[12] M. Koppitz *et al.*, "Organic Solar Modules: Fully Doctor Bladed on Glass in Air," *Energy Technology*, vol. 5, no. 7, pp. 1105–1111, Jul. 2017.

[13] J. You *et al.*, "A polymer tandem solar cell with 10.6% power conversion efficiency," *Nature Communications*, vol. 4, no. 1, p. 1446, Jun. 2013.

[14] W. Zhao *et al.*, "Fullerene-Free Polymer Solar Cells with over 11% Efficiency and Excellent Thermal Stability," *Advanced Materials*, vol. 28, no. 23, pp. 4734–4739, Jun. 2016.

[15] J.-D. Chen *et al.*, "Single-Junction Polymer Solar Cells Exceeding 10% Power Conversion Efficiency," *Advanced Materials*, vol. 27, no. 6, pp. 1035–1041, Feb. 2015.

[16] K. H. Park, Y. An, S. Jung, H. Park, and C. Yang, "The use of an n-type macromolecular additive as a simple yet effective tool for improving and stabilizing the performance of organic solar cells," *Energy & Environmental Science*, vol. 9, no. 11, pp. 3464–3471, Nov. 2016.

[17] M. C. Scharber *et al.*, "Design Rules for Donors in Bulk-Heterojunction Solar Cells—Towards 10 % Energy-Conversion Efficiency," *Advanced Materials*, vol. 18, no. 6, pp. 789–794, Mar. 2006.

[18] C. J. Brabec, S. E. Shaheen, C. Winder, N. S. Sariciftci, and P. Denk, "Effect of LiF/metal electrodes on the performance of plastic solar cells," *Applied Physics Letters*, vol. 80, no. 7, pp. 1288–1290, Feb. 2002.

[19] T. M. Brown, J. S. Kim, R. H. Friend, F. Cacialli, R. Daik, and W. J. Feast, "Built-in field electroabsorption spectroscopy of polymer light-emitting diodes incorporating a doped poly(3,4-ethylene dioxythiophene) hole injection layer," *Applied Physics Letters*, vol. 75, no. 12, pp. 1679–1681, Sep. 1999.

[20] M. P. de Jong, L. J. van IJzendoorn, and M. J. A. de Voigt, "Stability of the interface between indium-tin-oxide and poly(3,4-ethylenedioxythiophene)/poly(styrenesulfonate) in polymer light-emitting diodes," *Applied Physics Letters*, vol. 77, no. 14, pp. 2255–2257, Oct. 2000.

[21] C. T. Howells *et al.*, "Influence of perfluorinated ionomer in PEDOT:PSS on the rectification and degradation of organic photovoltaic cells," *Journal of Materials Chemistry A*, vol. 6, no. 33, pp. 16012–16028, 2018.

[22] G. Chamberlain, "Organic solar cells: A review," *Solar Cells*, vol. 8, no. 1, pp. 47–83, Feb. 1983.

[23] D. Wöhrle and D. Meissner, "Organic Solar Cells," *Advanced Materials*, vol. 3, no. 3, pp. 129–138, Mar. 1991.

[24] C. W. Tang, "Two-layer organic photovoltaic cell," *Applied Physics Letters*, vol. 48, no. 2, pp. 183–185, Jan. 1986.

[25] G. Yu, J. Gao, J. C. Hummelen, F. Wudl, and A. J. Heeger, "Polymer Photovoltaic Cells: Enhanced Efficiencies via a Network of Internal Donor-Acceptor Heterojunctions," *Science (New York, N.Y.)*, vol. 270,

no. 5243, pp. 1789–1791, 1995.

[26] C. Yang and A. Heeger, "Morphology of composites of semiconducting polymers mixed with C60," *Synthetic Metals*, vol. 83, no. 2, pp. 85–88, Nov. 1996.

[27] F. Padinger, R. Rittberger, and N. Sariciftci, "Effects of Postproduction Treatment on Plastic Solar Cells," *Advanced Functional Materials*, vol. 13, no. 1, pp. 85–88, Jan. 2003.

[28] R. Suliman, A. F. Mitul, L. Mohammad, G. Djira, Y. Pan, and Q. Qiao, "Modeling of organic solar cell using response surface methodology," *Results in Physics*, vol. 7, pp. 2232–2241, Jan. 2017.

[29] D. S. Pillai, B. Sahoo, J. P. Ram, A. Laudani, N. Rajasekar, and N. Sudhakar, "Modellingof Organic Photovoltaic Cells Based on an Improved Reverse Double Diode Model," *Energy Procedia*, vol. 117, pp. 1054–1061, Jun. 2017.

[30] A. H. Fallahpour *et al.*, "Modeling and simulation of energetically disordered organic solar cells," *Journal of Applied Physics*, vol. 116, no. 18, 2014.

[31] L. Hsu and W. Walukiewicz, "Modeling of InGaN/Si tandem solar cells," *Journal of Applied Physics*, vol. 104, no. 2, p. 024507, Jul. 2008.

[32] S. S. Haykin and Simon, *Neural networks : a comprehensive foundation.* Prentice Hall, 1999, p. 842.

A Appendix

A.1 Utitlity MLP Code

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 22 21:11:45 2019

@author: Depp Pc
"""
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import random


def plot_true_vs_pred(ytest, yhat, ytrain, ytrain_out, norm_lims, exp_str,
                      j,which_conc_feature, optimizer = ""):
    ytest_df=pd.DataFrame({'target'+ which_conc_feature[7:]:ytest})
    yhat_df=pd.DataFrame({'target'+ which_conc_feature[7:]:yhat})
    ytrain_df=pd.DataFrame({'target'+ which_conc_feature[7:]:ytrain})
    ytrain_out_df=pd.DataFrame({'target'+ which_conc_feature[7:]:ytrain_out})

    ytest_unnormed=make_unnorm(ytest_df,norm_lims)
    yhat_unnormed=make_unnorm(yhat_df,norm_lims)
    ytrain_unnormed=make_unnorm(ytrain_df,norm_lims)
    ytrain_out_unnormed=make_unnorm(ytrain_out_df,norm_lims)

    plt.plot(ytest_unnormed, 'bs', label='True Value')
    plt.plot(yhat_unnormed, 'r^', label='Predicted Value')
    plt.legend()
   # t = np.arange(0, len(ytest_unnormed), 1)
    #plt.xticks(t)
    plt.grid(which='both', axis = 'both')
    plt.xlabel('Test Samples',fontsize=20)
    plt.ylabel('T%', fontsize=20)
    plt.suptitle('Transmittance Prediction of Test Samples',fontsize=20)
    plt.savefig(fname=exp_str + "/figures/test_learned_nh_" + str(j) +
                which_conc_feature + optimizer + ".pdf")
    plt.show()
```

```
    plt.plot(ytrain_unnormed, 'bs', label='True Value')
    plt.plot(ytrain_out_unnormed, 'r^',label='Predicted Value')
    plt.legend()
   # t = np.arange(0, len(ytrain_unnormed), 1)
    #plt.xticks(t)
    plt.grid(which='both', axis = 'both')
    plt.xlabel('Training Samples',fontsize=20)
    plt.ylabel('T%', fontsize=20)
    plt.suptitle('Transmittance Prediction of Training Samples',fontsize=15)
    plt.savefig(fname=exp_str + "/figures/train_learned_nh_" + str(j)
    + which_conc_feature + optimizer + ".pdf")
    plt.show()


def make_norm(data,*restcol):
    from copy import deepcopy
    data_n = deepcopy(data)

    norm_limits = {}
    for colname in restcol:
        meanthis= (np.mean(data[colname]))
        stdthis= (np.std(data[colname]))
        norm_limits[colname] = {'mean': meanthis, 'std': stdthis}
        data_n[colname] = (data[colname] - meanthis)/stdthis
#         maxthis = max(data[colname])
#         minthis = min(data[colname])
        #norm_limits[colname] = {'max':maxthis, 'min':minthis}
        #data_n[colname]=(2*((data[colname]-minthis)/(maxthis-minthis))) - 1;
    return data_n, norm_limits

def make_norm_given_lim(data, norm_lims, *restcol):
    from copy import deepcopy
    data_n = deepcopy(data)
```

```
    for colname in norm_lims.keys():
        if colname not in data.columns.tolist():
            continue
        else:
            data_n[colname] = (data[colname] -
                  norm_lims[colname]['mean'])/norm_lims[colname]['std']
    return data_n


def make_unnorm(data, normlims):
    from copy import deepcopy
    data_unnorm = deepcopy(data)

    for colname in normlims.keys():
        if colname not in data.columns.tolist():
            continue
        else:
            data_unnorm[colname] = data[colname]*normlims[colname]['std']
            + normlims[colname]['mean']

    return data_unnorm


def my_r2_score(v_true, v_pred):
    ssres = np.sum(np.square(v_true - v_pred))
    sstot = np.sum(np.square(v_true - np.mean(v_true)))
    return 1 - ssres / sstot




def split_tt(df_normed, train_percent,iteration=3, *featurename):
    n=train_percent/100
    dftrain = df_normed.sample(frac =n, random_state=iteration)
```

```
dftrain.sort_index(inplace=True)
dftest=df_normed.drop(dftrain.index)
actual_perc=(len(dftrain)/len(df_normed))*100
print(actual_perc)
tempp=[]
for j in featurename:
    temp_list = []
    for k in featurename:
        temp_list.append(k)
    tempp.append(temp_list)

for j in temp_list:
    if j[0:7] == 'feature':
        target_name= 'target' + j[7:]
    else:
        target_name= 'target'
xdftrain=dftrain
ydftrain=dftrain
xdftest=dftest
ydftest=dftest
xdftrain=xdftrain.filter(items=featurename, axis=1)
ydftrain=ydftrain.filter(items=[target_name], axis=1)
xdftest=xdftest.filter(items=featurename,axis=1)
ydftest=ydftest.filter(items=[target_name],axis=1)
xdftrain=xdftrain.dropna(axis=1)
ydftrain=ydftrain.dropna(axis=1)
xdftest=xdftest.dropna(axis=1)
ydftest=ydftest.dropna(axis=1)
xtrain = xdftrain.values
xtrain=xtrain.reshape((len(xtrain),len(featurename)))
ytrain = ydftrain.values
ytrain=ytrain.reshape(len(ytrain),)
xtest = xdftest.values
xtest=xtest.reshape((len(xtest),len(featurename)))
ytest = ydftest.values
ytest=ytest.reshape(len(ytest),)
return xdftrain, ydftrain, xdftest, ydftest, xtrain, ytrain, xtest, ytest,
```

A.2 Utitlity FLANN Code

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 22 21:11:45 2019

@author: Depp Pc
"""
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import random


def plot_true_vs_pred(ytest, yhat, ytrain, ytrain_out, norm_lims, str_path,
                      exp_str, j,which_conc_feature, optimizer = ""):
    ytest_df=pd.DataFrame({'target':ytest})
    yhat_df=pd.DataFrame({'target':yhat})
    ytrain_df=pd.DataFrame({'target':ytrain})
    ytrain_out_df=pd.DataFrame({'target':ytrain_out})

    ytest_unnormed=make_unnorm(ytest_df,norm_lims)
    yhat_unnormed=make_unnorm(yhat_df,norm_lims)
    ytrain_unnormed=make_unnorm(ytrain_df,norm_lims)
    ytrain_out_unnormed=make_unnorm(ytrain_out_df,norm_lims)

    plt.plot(ytest_unnormed, 'bs', label='True Value')
    plt.plot(yhat_unnormed, 'r^', label='Predicted Value')
    plt.legend()

    plt.grid(which='both', axis = 'both')
    plt.xlabel('Test Samples',fontsize=20)
    plt.ylabel('T%', fontsize=20)
    plt.suptitle('Transmittance Prediction of Test Samples',fontsize=20)
    plt.savefig(fname=str_path + "/"+ exp_str + "/figures/test_learned_nh_"
                + str(j) + which_conc_feature + optimizer + ".pdf")
    plt.show()
```

```
    plt.plot(ytrain_unnormed, 'bs', label='True Value')
    plt.plot(ytrain_out_unnormed, 'r^',label='Predicted Value')
    plt.legend()

    plt.grid(which='both', axis = 'both')
    plt.xlabel('Training Samples',fontsize=20)
    plt.ylabel('T%', fontsize=20)
    plt.suptitle('Transmittance Prediction of Training Samples',fontsize=20)
    plt.savefig(fname=str_path + "/"+ exp_str + "/figures/train_learned_nh_"
                + str(j) + which_conc_feature + optimizer + ".pdf")
    plt.show()

def make_norm(data,*restcol):
    from copy import deepcopy
    data_n = deepcopy(data)

    norm_limits = {}
    for colname in restcol:
        meanthis= (np.mean(data[colname]))
        stdthis= (np.std(data[colname]))
        norm_limits[colname] = {'mean': meanthis, 'std': stdthis}
        data_n[colname] = (data[colname] - meanthis)/stdthis

    return data_n, norm_limits

def make_norm_given_lim(data, norm_lims, *restcol):
    from copy import deepcopy
    data_n = deepcopy(data)

    for colname in norm_lims.keys():
        if colname not in data.columns.tolist():
            continue
        else:
            data_n[colname] = (data[colname] -
                  norm_lims[colname]['mean'])/norm_lims[colname]['std']
    return data_n
```

```
def make_unnorm(data, normlims):
    from copy import deepcopy
    data_unnorm = deepcopy(data)

    for colname in normlims.keys():
        if colname not in data.columns.tolist():
            continue
        else:
            data_unnorm[colname] =(data[colname] +
                       normlims[colname]['std']) + normlims[colname]['mean']
    return data_unnorm

def my_r2_score(v_true, v_pred):
    ssres = np.sum(np.square(v_true - v_pred))
    sstot = np.sum(np.square(v_true - np.mean(v_true)))
    return 1 - ssres / sstot

def split_tt(df_normed, train_percent,iteration=3, *featurename):
    n=train_percent/100
    dftrain = df_normed.sample(frac =n, random_state=iteration)
    dftrain.sort_index(inplace=True)
    dftest=df_normed.drop(dftrain.index)
    actual_perc=(len(dftrain)/len(df_normed))*100
    print(actual_perc)
    tempp=[]
    for j in featurename:
        temp_list = []
        for k in featurename:
            temp_list.append(k)
        tempp.append(temp_list)

    for j in temp_list:
        if j[0:7] == 'feature':
            target_name= 'target' + j[7:]
        elif j[0:10] == 'wavelength' and len(j)>10:
            continue
        else:
            target_name= 'target'
```

```
xdftrain=dftrain
ydftrain=dftrain
xdftest=dftest
ydftest=dftest
xdftrain=xdftrain.filter(items=featurename, axis=1)
ydftrain=ydftrain.filter(like=target_name, axis=1)
xdftest=xdftest.filter(items=featurename,axis=1)
ydftest=ydftest.filter(like=target_name,axis=1)
xdftrain=xdftrain.dropna(axis=1)
ydftrain=ydftrain.dropna(axis=1)
xdftest=xdftest.dropna(axis=1)
ydftest=ydftest.dropna(axis=1)
xtrain = xdftrain.values
xtrain=xtrain.reshape((len(xtrain),len(featurename)))
ytrain = ydftrain.values
ytrain=ytrain.reshape(len(ytrain),)
xtest = xdftest.values
xtest=xtest.reshape((len(xtest),len(featurename)))
ytest = ydftest.values
ytest=ytest.reshape(len(ytest),)
return xdftrain, ydftrain, xdftest, ydftest, xtrain, ytrain, xtest, ytest
```

A.3 MLP Code

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 28 10:48:24 2019

@author: Depp Pc
"""


import os
exp_str = "dataout/exp1_1"
os.makedirs(exp_str+"/figures", exist_ok=True)
os.chdir("D:\python\SampleANN\Transmittance")
import pandas as pd
import numpy as np
import time
import scipy.stats
from numpy.random import seed
from tensorflow import set_random_seed

from utility import make_norm, make_unnorm, my_r2_score, split_tt,
plot_true_vs_pred
#Read data file
df = pd.read_csv("transmittance.csv")
import math
df_normed, norm_lims = make_norm(df, *df.columns.tolist())


which_conc_feature = "feature13_4"

xdftrain, ydftrain, xdftest, ydftest, xtrain, ytrain, xtest, ytest,
= split_tt(df_normed,70,102,which_conc_feature)



print("Shape of Training feature matrix as array: ",np.shape(xtrain))
print("Shape of Target array: ",np.shape(ytrain))
print("Dimension of training data array: ",np.ndim(xtrain))
```

```
print("Dimension of target array: ",np.ndim(ytrain))

from keras.models import Sequential
from keras.layers import Dense, Activation
import sklearn.metrics as skm
dim_output = 1
dim_input = 1
len_train_input = ytrain.shape[0]
performance_out=pd.DataFrame()
for j in range(3,20):
    num_hidden_neuron= j

    xdftrain, ydftrain,xdftest, ydftest, xtrain, ytrain, xtest, ytest,
    = split_tt(df_normed,70,102, which_conc_feature)

    mymodel = Sequential([
            Dense(units = num_hidden_neuron, input_dim = dim_input,
                  activation='tanh'),
            Dense(1, activation='linear'),
    ])

    mymodel.compile(optimizer='Adam',
              loss='mse')


    #Fit the model to our training data
        #-------------seed-----------------
    seed(102)
    set_random_seed(102)
    #-----------------------------------
    start=time.time()
    myhist = mymodel.fit(xtrain,ytrain, batch_size=len_train_input,
                         epochs=500, verbose=0)
    end=time.time()
    train_time= [end-start]
```

v

```
#Predict target values with our test data
yhat = mymodel.predict(x=xtest)
yhat = yhat.reshape(len(yhat))

ytrain_out = mymodel.predict(x=xtrain)
ytrain_out = ytrain_out.reshape(len(ytrain_out))


entire_predicted= np.concatenate((ytrain_out,yhat))
entire_true = np.concatenate((ytrain,ytest))

#Calculate train, test and overall mean squared error
test_mse = [skm.mean_squared_error(ytest,yhat)]
train_mse= [skm.mean_squared_error(ytrain,ytrain_out)]
overall_mse= [skm.mean_squared_error(entire_true,entire_predicted)]

test_mae= [skm.mean_absolute_error(ytest,yhat)]

myRsq = [my_r2_score(v_true=ytest, v_pred=yhat)]
myRsq_training = [my_r2_score(v_true=ytrain, v_pred=ytrain_out)]

no_iter=myhist.epoch[-1]+1
useless_modelcount=0
useless_modelcount_training=0

if myRsq[0] < 0:
    useless_modelcount= useless_modelcount + 1
if myRsq_training[0] < 0:
    useless_modelcount_training= useless_modelcount_training + 1

n_ensemble=100
for i in range(n_ensemble-1):
    xdftrain, ydftrain,xdftest, ydftest, xtrain, ytrain, xtest, ytest,
    = split_tt(df_normed,70,i+1,which_conc_feature)
    #-------------seed------------------
    seed(i+1)
    set_random_seed(i+1)
```

```
#-------------------------------------
start=time.time()
myhist = mymodel.fit(xtrain,ytrain, batch_size=len_train_input,
                     epochs=500, verbose=0)
end=time.time()
print("Training time is:", round(end-start,3))
train_time= train_time + [(end-start)]
no_iter = no_iter + (myhist.epoch[-1]+1)
print("The number of iterations ran was: ",myhist.epoch[-1]+1)
yhat = mymodel.predict(xtest); yhat = yhat.reshape(len(yhat))
ytrain_out= mymodel.predict(xtrain); ytrain_out =
ytrain_out.reshape(len(ytrain_out))
entire_predicted= np.concatenate((ytrain_out,yhat))
entire_true = np.concatenate((ytrain,ytest))
#performance metrics
    #MSE
test_mse= test_mse + [skm.mean_squared_error(ytest,yhat)]
train_mse= train_mse + [skm.mean_squared_error(ytrain,ytrain_out)]
overall_mse= overall_mse +
[skm.mean_squared_error(entire_true,entire_predicted)]
#MAE
test_mae= test_mae + [skm.mean_absolute_error(ytest,yhat)]
#R squared
current_Rsq = my_r2_score(v_pred=yhat, v_true=ytest)
current_Rsq_training = my_r2_score(v_pred=ytrain_out, v_true=ytrain)

#if (current_Rsq_training < 0):
    #import pdb; pdb.set_trace()
myRsq = myRsq + [current_Rsq]
myRsq_training = myRsq_training + [current_Rsq_training]

if current_Rsq < 0:
    useless_modelcount= useless_modelcount + 1
if current_Rsq_training < 0:
    useless_modelcount_training= useless_modelcount_training + 1
```

```
# calculate means and standard error of the means of the various measures
avg_train_time_mean = np.mean(train_time)
avg_train_time_sem = scipy.stats.sem(train_time)
test_fmse_mean = np.mean(test_mse)    #find average mse over 100 ensembles
test_fmse_sem = scipy.stats.sem(test_mse)
train_fmse_mean = np.mean(train_mse)
train_fmse_sem = scipy.stats.sem(train_mse)

overall_fmse_mean = np.mean(overall_mse)
overall_fmse_sem = scipy.stats.sem(overall_mse)
test_fmae_mean = np.mean(test_mae)
test_fmae_sem = scipy.stats.sem(test_mae)
my_fRsq_mean = np.mean(myRsq)
my_fRsq_sem = scipy.stats.sem(myRsq)
my_fRsq_training_mean = np.mean(myRsq_training)
my_fRsq_training_sem = scipy.stats.sem(myRsq_training)

overall_MSEdB= round(math.log10(overall_fmse_mean),3)
test_MSEdB= round(math.log10(test_fmse_mean),3)
train_MSEdB= round(math.log10(train_fmse_mean),3)

print("")
print("number of models with negative R squared (test data)", useless_modelcount)
print("")
print("number of models with negative R squared (training data)", useless_modelcount_training)
print("")
print("average mean squared error of entire dataset",
      round(math.log10(overall_fmse_mean),3),"dB")
print ("average mean squared error of testing set is: ",
       round(math.log10(test_fmse_mean),3),"dB")
print("average mean squared error for training set",
      round(math.log10(train_fmse_mean),3),"dB")
print ("average mean absolute error of testing set is: ",
       round(math.log10(test_fmae_mean),3),"dB")
print("average R squared (test data) for the model is: ",
      round(my_fRsq_mean,4))
```

```
print("")
print("average R squared (training data) for the model is: ",
      round(my_fRsq_training_mean,4))
print("")

no_fiter=no_iter/n_ensemble
print("avg iterations",no_fiter)
print("")
# intialise data of lists.
temp = pd.DataFrame({'Nh':j, 'Overall dB': overall_MSEdB,
                     'Test dB': test_MSEdB, 'Train dB': train_MSEdB,
                     'Overall MSE mean': overall_fmse_mean,
                     'Test MSE mean': test_fmse_mean,
                     'Train MSE mean': train_fmse_mean,
                     'Overall MSE sem': overall_fmse_sem,
                     'Test MSE sem': test_fmse_sem,
                     'Train MSE sem': train_fmse_sem,
                     'Avg. R Squared (test) mean': my_fRsq_mean,
                     'Avg. R Squared (training) mean': my_fRsq_training_mean,
                     'Avg. R Squared (test) sem': my_fRsq_sem,
                     'Avg. R Squared (training) sem': my_fRsq_training_sem,
                     'Avg. Train Time mean':avg_train_time_mean,
                     'Avg. Train Time sem':avg_train_time_sem,
                     'Avg. no of iterations':no_fiter,
                     'No. of useless models (test)': useless_modelcount,
                     'No. of useless models (training)':useless_modelcount_training},
index=[0])
performance_out = pd.concat([performance_out, temp])

import pickle

pklfile = open(file=exp_str + "/train_learned_nh_" + str(num_hidden_neuron)
+ which_conc_feature + ".pkl", mode="wb")
toPickle = {'xtrain': xtrain, 'ytrain':ytrain , 'xtest':xtest ,
            'ytest':ytest ,'yhat':yhat, 'entire_predicted':entire_predicted
            , 'entire_true': entire_true,'norm_lims' : norm_lims,
            'test_mse':test_mse , 'train_mse':train_mse ,
```

```
                'overall_mse':overall_mse , 'test_mae': test_mae,
                'current_Rsq': current_Rsq ,
                'current_Rsq_training':current_Rsq_training,
                'loss_iter':myhist.history['loss'] }
    pickle.dump(obj = toPickle, file=pklfile)
    pklfile.close()

    perror = 100.0*(np.abs(ytest-yhat)/ytest)
    #find maximum percentage error
    maxperror = np.max(100.0*(np.abs(ytest-yhat)/ytest))
    print("Maximum percet error = ",maxperror)
    plot_true_vs_pred(ytest,yhat,ytrain,ytrain_out, norm_lims,exp_str,j,
                      which_conc_feature)

export_csv = performance_out.to_csv(exp_str +"/"+ r'Ynew_performances_1_1.csv',
                                    index = None, header=True) #

pklfile = open(file=exp_str + "/train_learned_nh_" + str(num_hidden_neuron) +
               which_conc_feature+".pkl", mode="rb")
obj = pickle.load(pklfile)
pklfile.close()
```

A.4 FLANN Code

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 28 10:48:24 2019

@author: Depp Pc
"""
import os
exp_str = "dataout/exp1_1"

os.chdir("D:\python\SampleANN\Transmittance_FLANN")
import pandas as pd
import numpy as np
import time
import scipy.stats
from numpy.random import seed
from tensorflow import set_random_seed

from utility_flann import make_norm, make_unnorm, my_r2_score, split_tt,
 plot_true_vs_pred
#Read data file
df = pd.read_csv("transmittance.csv")
import math

df_normed, norm_lims = make_norm(df, *df.columns.tolist())
which_conc_feature= "feature13_4"
concentration= which_conc_feature[7:]
df_normed=df_normed.loc[:, df_normed.columns.str.endswith(concentration)]

def which_flann(df, flann_polynomial):
    if flann_polynomial=='chebyshev':
        str_path="chebyshev"
        df['wavelength2']=1
        df['wavelength3']=2*df[which_conc_feature]**2-1
        df['wavelength4']=2*df[which_conc_feature]**3-3*df[which_conc_feature]
        df['wavelength5']=8*df[which_conc_feature]**4-8*df[which_conc_feature]**2+1
```

```
    if flann_polynomial=='legendre':
        str_path="legendre"
        df['wavelength2']=1
        df['wavelength3']=(3*df[which_conc_feature]**2-1)/2
        df['wavelength4']=(5*df[which_conc_feature]**3-3*df[which_conc_feature])/2
        df['wavelength5']=(35*df[which_conc_feature]**4-30*df[which_conc_feature]**2+3)/8


    if flann_polynomial=='trigonometric':
        str_path="trigonometric"
        df['wavelength2']=np.cos(df[which_conc_feature])
        df['wavelength3']=np.sin(df[which_conc_feature])
        df['wavelength4']=np.cos(2*math.pi*df[which_conc_feature])
        df['wavelength5']=np.sin(2*math.pi*df[which_conc_feature])

    if flann_polynomial=='power':
        str_path="power"
        df['wavelength2']=1
        df['wavelength3']=df[which_conc_feature]**2
        df['wavelength4']=df[which_conc_feature]**3
        df['wavelength5']=df[which_conc_feature]**4

    else:
        print("invalid flann. check spelling")
    return str_path, df


str_path, df_normed_expanded = which_flann(df_normed, 'chebyshev')
1
os.makedirs(str_path + "/"+ exp_str +"/figures", exist_ok=True)

feature_list=[which_conc_feature ,'wavelength2','wavelength3','wavelength4','wavelength5']
xdftrain, ydftrain, xdftest, ydftest, xtrain, ytrain, xtest, ytest  =
split_tt(df_normed_expanded,70,102, *feature_list)
```

```
print("Shape of Training feature matrix as array: ",np.shape(xtrain))
print("Shape of Target array: ",np.shape(ytrain))
print("Dimension of training data array: ",np.ndim(xtrain))
print("Dimension of target array: ",np.ndim(ytrain))

from keras.models import Sequential
from keras.layers import Dense, Activation
import sklearn.metrics as skm


dim_output = 1
dim_input = 5
len_train_input = ytrain.shape[0]


performance_out=pd.DataFrame()

for j in range(1):
    num_hidden_neuron= j+1
    #make a model with specified hyper parameters
    mymodel = Sequential([
            Dense(units = num_hidden_neuron, input_dim = dim_input, activation='tanh'),
            Dense(1, activation='linear'),
    ])

    mymodel.compile(optimizer='RMSprop',
              loss='mse')

    xdftrain, ydftrain, xdftest, ydftest, xtrain, ytrain, xtest, ytest
    = split_tt(df_normed_expanded,70,102,*feature_list)
    #-------------seed------------------
    seed(102)
    set_random_seed(102)
    #-----------------------------------
```

```
#Fit the model to our training data
start=time.time()
myhist = mymodel.fit(xtrain,ytrain, batch_size=len_train_input,
                     epochs=500, verbose=0)
end=time.time()
train_time= [end-start]



#Predict target values with our test data
yhat = mymodel.predict(x=xtest)
yhat = yhat.reshape(len(yhat))

ytrain_out = mymodel.predict(x=xtrain)
ytrain_out = ytrain_out.reshape(len(ytrain_out))

entire_predicted= np.concatenate((ytrain_out,yhat))
entire_true = np.concatenate((ytrain,ytest))

#Calculate train, test and overall mean squared error
test_mse = [skm.mean_squared_error(ytest,yhat)]
train_mse= [skm.mean_squared_error(ytrain,ytrain_out)]
overall_mse= [skm.mean_squared_error(entire_true,entire_predicted)]

test_mae= [skm.mean_absolute_error(ytest,yhat)]

myRsq = [my_r2_score(v_true=ytest, v_pred=yhat)]
myRsq_training = [my_r2_score(v_true=ytrain, v_pred=ytrain_out)]

no_iter=myhist.epoch[-1]+1
useless_modelcount=0
useless_modelcount_training=0

if myRsq[0] < 0:
    useless_modelcount= useless_modelcount + 1
```

```
if myRsq_training[0] < 0:
    useless_modelcount_training= useless_modelcount_training + 1

n_ensemble=100
for i in range(n_ensemble-1):
    xdftrain, ydftrain, xdftest, ydftest, xtrain, ytrain, xtest, ytest =
    split_tt(df_normed_expanded,70,102, *feature_list)
    #-------------seed------------------
    seed(i+1)
    set_random_seed(i+1)
    #-----------------------------------
    start=time.time()
    myhist = mymodel.fit(xtrain,ytrain, batch_size=len_train_input, epochs=500, verbose=0)
    end=time.time()
    print("Training time is:", round(end-start,3))
    train_time= train_time + [(end-start)]
    no_iter = no_iter + (myhist.epoch[-1]+1)
    print("The number of iterations ran was: ",myhist.epoch[-1]+1)
    yhat = mymodel.predict(xtest); yhat = yhat.reshape(len(yhat))
    ytrain_out= mymodel.predict(xtrain); ytrain_out = ytrain_out.reshape(len(ytrain_out))
    entire_predicted= np.concatenate((ytrain_out,yhat))
    entire_true = np.concatenate((ytrain,ytest))
    #performance metrics
        #MSE
    test_mse= test_mse + [skm.mean_squared_error(ytest,yhat)]
    train_mse= train_mse + [skm.mean_squared_error(ytrain,ytrain_out)]
    overall_mse= overall_mse + [skm.mean_squared_error(entire_true,entire_predicted)]
    #MAE
    test_mae= test_mae + [skm.mean_absolute_error(ytest,yhat)]
    #R squared
    current_Rsq = my_r2_score(v_pred=yhat, v_true=ytest)
    current_Rsq_training = my_r2_score(v_pred=ytrain_out, v_true=ytrain)

    if (current_Rsq_training < 0):
        import pdb; pdb.set_trace()
    myRsq = myRsq + [current_Rsq]
    myRsq_training = myRsq_training + [current_Rsq_training]
```

```
    if current_Rsq < 0:
        useless_modelcount= useless_modelcount + 1
    if current_Rsq_training < 0:
        useless_modelcount_training= useless_modelcount_training + 1


avg_train_time_mean = np.mean(train_time)
avg_train_time_sem = scipy.stats.sem(train_time)


test_fmse_mean = np.mean(test_mse)
test_fmse_sem = scipy.stats.sem(test_mse)
train_fmse_mean = np.mean(train_mse)
train_fmse_sem = scipy.stats.sem(train_mse)

overall_fmse_mean = np.mean(overall_mse)
overall_fmse_sem = scipy.stats.sem(overall_mse)

test_fmae_mean = np.mean(test_mae)
test_fmae_sem = scipy.stats.sem(test_mae)

my_fRsq_mean = np.mean(myRsq)
my_fRsq_sem = scipy.stats.sem(myRsq)

my_fRsq_training_mean = np.mean(myRsq_training)
my_fRsq_training_sem = scipy.stats.sem(myRsq_training)


overall_MSEdB= round(10*math.log10(overall_fmse_mean),3)
test_MSEdB= round(10*math.log10(test_fmse_mean),3)
train_MSEdB= round(10*math.log10(train_fmse_mean),3)

print("")
print("number of models with negative R squared (test data)", useless_modelcount)
print("")
```

```
print("number of models with negative R squared (training data)", useless_modelcount_training)
print("")
print("average mean squared error of entire dataset",
      round(10*math.log10(overall_fmse_mean),3),"dB")
print ("average mean squared error of testing set is: ",
       round(10*math.log10(test_fmse_mean),3),"dB")
print("average mean squared error for training set",
      round(10*math.log10(train_fmse_mean),3),"dB")

print ("average mean absolute error of testing set is: ",
       round(10*math.log10(test_fmae_mean),3),"dB")

print("average R squared (test data) for the model is: ",
      round(my_fRsq_mean,4))
print("")

print("average R squared (training data) for the model is: ",
      round(my_fRsq_training_mean,4))
print("")

no_fiter=no_iter/n_ensemble
print("avg iterations",no_fiter)
print("")
# intialise data of lists.
temp = pd.DataFrame({'Nh':j, 'Overall dB': overall_MSEdB,
                     'Test dB': test_MSEdB, 'Train dB': train_MSEdB,
                     'Overall MSE mean': overall_fmse_mean,
                     'Test MSE mean': test_fmse_mean,
                     'Train MSE mean': train_fmse_mean,
                     'Overall MSE sem': overall_fmse_sem,
                     'Test MSE sem': test_fmse_sem,
                     'Train MSE sem': train_fmse_sem,
                     'Avg. R Squared (test) mean': my_fRsq_mean,
                     'Avg. R Squared (training) mean': my_fRsq_training_mean,
                     'Avg. R Squared (test) sem': my_fRsq_sem,
                     'Avg. R Squared (training) sem': my_fRsq_training_sem,
```

```
                        'Avg. R Squared (training) mean': my_fRsq_training_mean,
                        'Avg. R Squared (test) sem': my_fRsq_sem,
                        'Avg. R Squared (training) sem': my_fRsq_training_sem,
                        'Avg. Train Time mean':avg_train_time_mean,
                        'Avg. Train Time sem':avg_train_time_sem,
                        'Avg. no of iterations':no_fiter,
                        'No. of useless models (test)': useless_modelcount,
                        'No. of useless models (training)': useless_modelcount_training},
    index=[0])
    performance_out = pd.concat([performance_out, temp])

    import pickle
    pklfile = open(file=str_path + "/"+ exp_str + "/" + which_conc_feature +
                   'FLANN_transmit_1_1.pkl', mode="wb")
    toPickle = {'xtrain': xtrain, 'ytrain':ytrain , 'xtest':xtest,
                'ytrain_out': ytrain_out, 'ytest':ytest , 'yhat':yhat,
                'entire_predicted':entire_predicted , 'entire_true': entire_true
                ,'norm_lims':norm_lims, 'test_mse':test_mse ,
                'train_mse':train_mse , 'overall_mse':overall_mse
                , 'test_mae': test_mae, 'current_Rsq': current_Rsq
                ,'current_Rsq_training':current_Rsq_training,
                'loss_iter':myhist.history['loss'] }
    pickle.dump(obj = toPickle, file=pklfile)
    pklfile.close()

    perror = 100.0*(np.abs(ytest-yhat)/ytest)
    #find maximum percentage error
    maxperror = np.max(100.0*(np.abs(ytest-yhat)/ytest))
    print("Maximum percet error = ",maxperror)

    plot_true_vs_pred(ytest,yhat,ytrain,ytrain_out, norm_lims,str_path,
                      exp_str,j,which_conc_feature, "")

os.makedirs(str_path + "/"+ exp_str, exist_ok=True)
export_csv = performance_out.to_csv(str_path + "/"+ exp_str +"/"+
                                    r'FLANN_performances_1_1.csv', index = None, header=True)
```