PROJECT REPORT

on

"ARTIFICIAL NEURAL NETWORK (ANN) MODELLING OF STYBLINSKI-TANG FUNCTION"

Submitted by

Anwesh Verma

EW22MTECH14001

COURSE NAME: Machine Learning for Process Systems Engineering (CH6870)

TABLE OF CONTENT:

1. INTRODUCTION

The styblinski–Tang function is a test function used to evaluate and test optimization Salgorithms. The function is given by

$$f(x_i) = \frac{\sum_i^n (x_i^4 - 16x_i^4 + 5x_i)}{2} \qquad (1)$$

where $x_i \epsilon (-5,5)$. 'n' represents the dimensions. For n=3, The number of independentvariables is (x_1, x_2, x_3). ANN modelling was performed to develop a functional relationship between the function value and these 3-inputs. The data for the model was generated using the function expression and it was used to build the model.

2. DATA GENERATION

I. Generation of Input and Target Data

The required packages such as ***pandas, NumPy, and random*** were imported into the code. The input data was randomly generated using the random function. The number of data points was taken as 4000. Hence the data was generated with the matrix space 4000 * 3. These generated data were given as input to the test function and its function value was evaluated.

```
import numpy as np
import pandas as pd
import random

dataset = 4000

variables = 3

X = np.random.uniform(-5.0, 5.0, size = (dataset, variables))


ST = []
for i in range (dataset):
    total = 0
    for j in range(variables):
        total = total+((X[i][j])**4 - 16*(X[i][j])**2+5*(X[i][j]))
        Total=total/2
    ST.append(Total)
```

II. Saving and Reading the from excel file

The generated data were then concatenated and it was assigned to a data frame variable using ***DataFrame***. These were then stored in an excel file locally.

```
data = pd.concat([pd.DataFrame(X),pd.DataFrame(ST)], axis=1)
data.to_excel('C:\\Users\\asus\\OneDrive\\Documents\\Anvesh_project\\datageneratio
n.xlsx')
```

The data from excel was read using ***read_excel*** and stored in an array for further processing.

```
Data=pd.read_excel('C:\\Users\\asus\\OneDrive\\Documents\\Anvesh_project\\datage
neration.xlsx')
```

III. Normalization of Data

The input and output data were normalized between 0 – 1. To normalize the data, ***MinMaxScaler*** tool was imported ***Skl.Preprocessing*** Package. Then the normalized data was stored for further usage.

```
data1=Data.iloc[:,1:5]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data1.values)
```

IV. Data for Training, Validation and Testing

The normalized data was split into input data and target data required for the development of the ANN model. 70% of the data was taken as the input data, 15% as validation and the rest 15 % as testing data.

Sr.No	Process	Data Percentage
1.	Training	70
2.	Validation	15
3.	Testing	15

```
x = scaled_data[:,0:3]
y = scaled_data[:,3]
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.15,
random_state=0)
```

3. ARTIFICIAL NEURAL NETWORK MODELLING

ANN modelling was implemented for the normalized data using ***TensorFlow*** package in python. ***Sequential and Dense*** functions were imported from TensorFlow and it was used to create the neural network.

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

I. Neural network using Adam optimizer

The network structure is as follows – **TABLE - 1**

Layer	Activation Function	No. of Nodes
Hidden Layer 1	LINEAR	90
Hidden Layer 2	RELU	100
Output Layer	RELU	1

*EPOCH = 500

```
net = Sequential()
net.add(Dense(90,input_dim=3,activation='linear'
net.add(Dense(100,activation='relu'))
net.add(Dense(1,activation='relu'))
```

The' ***adam'*** algorithm was taken as the optimizer and '***Mean Square Error (MSE)'*** was taken as the loss function.

```
net.compile(optimizer = 'adam', loss = 'MSE')

Network Summary:
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 90)                360

 dense_1 (Dense)             (None, 100)               9100

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 9,561
Trainable params: 9,561
Non-trainable params: 0
_________________________________________________________________
None
```

The number of iterations (epoch) was fixed as 500 and the validation data percentage was assigned as 15% of the dataset.

```
history = net.fit(x_train, y_train, epochs = 750, validation_split=0.15)
```

To validate the predicted results, statistical parameter like Mean Square Error (MSE), R^2 was imported from skllearn.metrics.

```
from sklearn.metrics import r2_score
r2_train = r2_score(y_train, y_train_pre)
r2_train
```

```
y_test_pre=net.predict(x_test)
r2_test = r2_score(y_test, y_test_pre)
r2_test
from sklearn.metrics import mean_squared_error
mse_train=mean_squared_error(y_train,y_train_pre)
mse_train
from sklearn.metrics import mean_squared_error
mse_test=mean_squared_error(y_test, y_test_pre)
mse_test
```

Sr. No.	**Predicted Data**	**MSE**	**R^2**
1.	Training	0.0004110212677971129	0.9805818796079804
2.	Testing	0.0004969570711123319	0.9788422368794105

The scatter plot of Predicted data vs True data was plotted using the scatter plot function imported from ***`matplotlib`***.

```
from matplotlib import pyplot as p
p.scatter(y_train, y_train_pre)
p.scatter(y_train, y_train_pre)
```

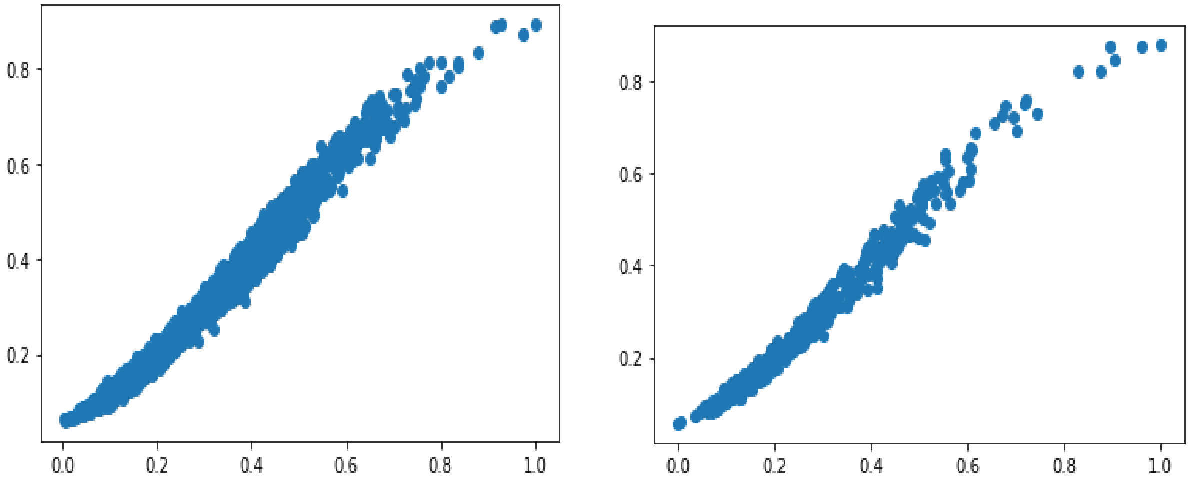


Figure 1: Predicted data Vs True data (Both Training and Testing) – *Adam Optimizer*

I. Neural network using RMSProp optimizer

The same network architecture was now trained with '***RMSProp***' as the optimizer and '***MSE'*** as the loss function.

```
net.compile(optimizer = 'RMSProp', loss = 'MSE')
```

Sr. No.	**Predicted Data**	**MSE**	**R^2**
1.	Training	0.00016484624046955119	0.9922120717480983

2.	Testing	0.0001556586617084596	0.9933728901679092

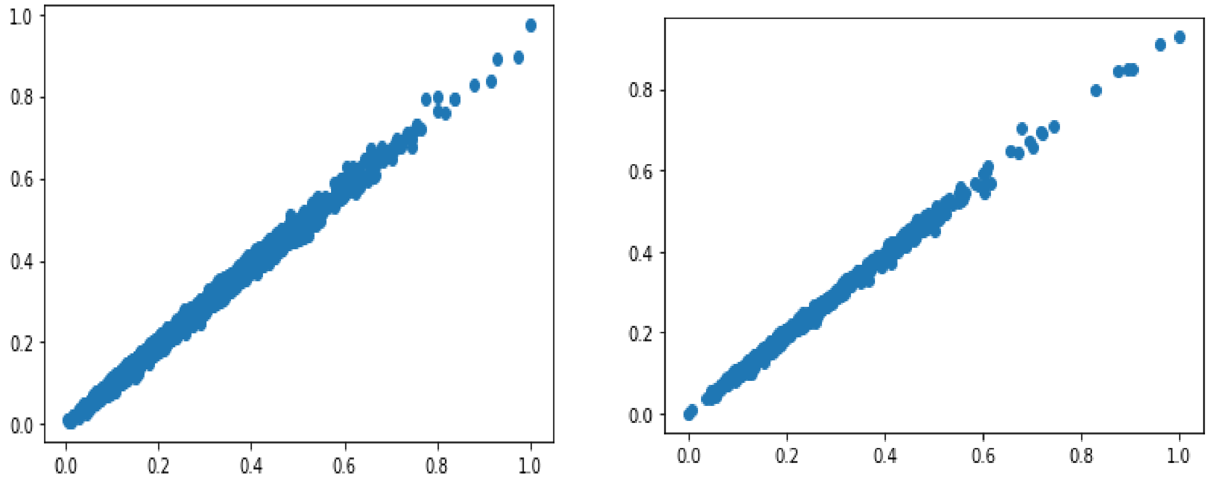


Figure 2: Predicted data Vs True data (Both Training and Testing) – *RMSProp optimizer*

4. RESULTS AND DISCUSSION

I. Effect of Epoch

To study the effect of the Number of Iteration (Epoch) on the prediction performance of the ANN model, the epoch was varied keeping the other parameters fixed as given in the network structure table 1. It was observed the 750 iterations gave the best performance in both the training and testing of the model. Hence for further analysis epoch was fixed as 750.

EPOCH	TRAINING		TESTING	
	R^2	MSE	R^2	MSE
250	0.9846823154953979	0.00032422778197344163	0.9843752833623037	0.0003669959519337558
500	0.9922120717480983	0.00016484624046955119	0.9933728901679092	0.0001556586617084596
750	0.9975712094412565	5.1409948775405295e-05	0.9963028957637146	8.683820129706703e-05
1000	0.9958217015670963	8.844159395739491e-05	0.9964443723902479	8.351517468257392e-05

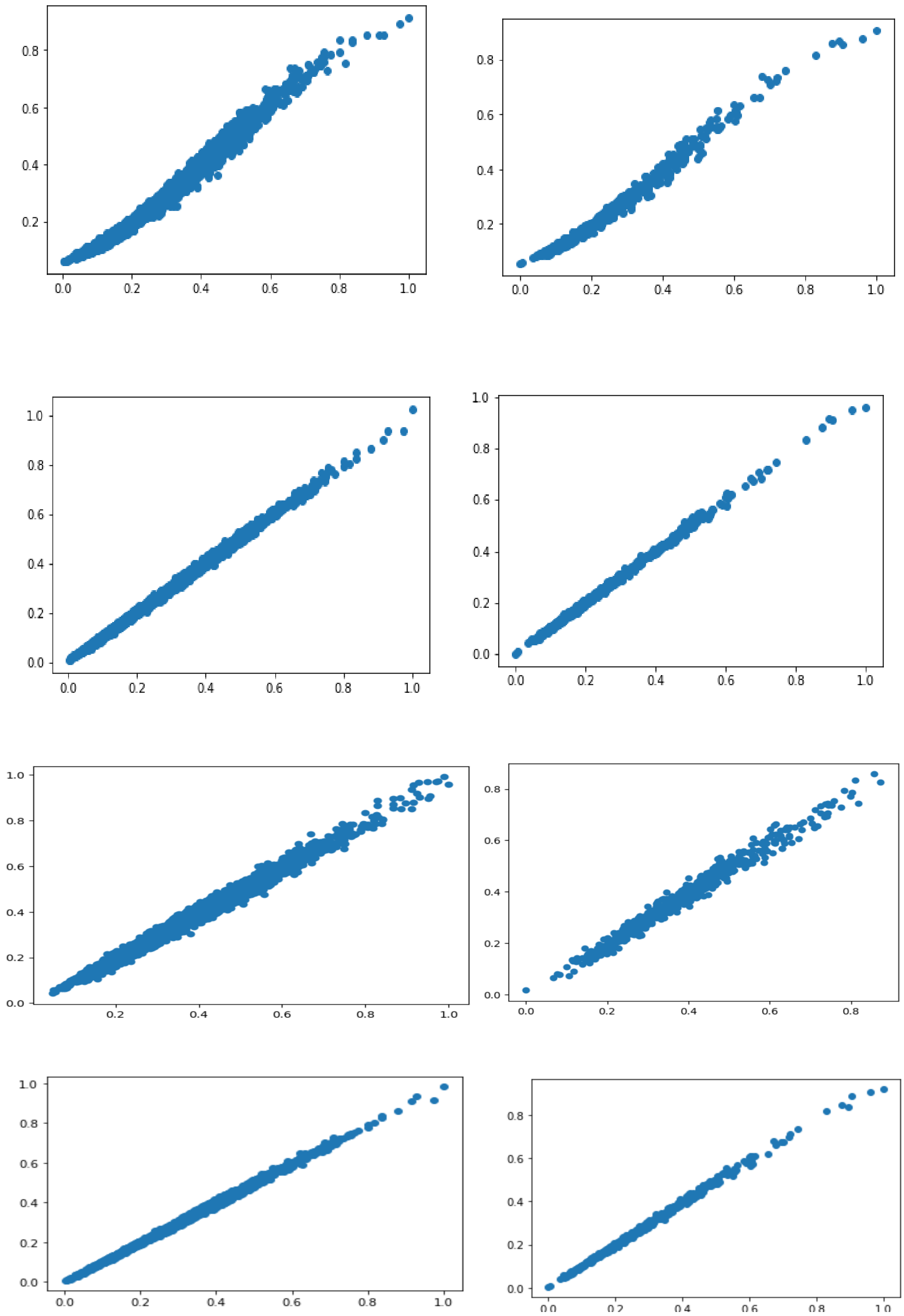
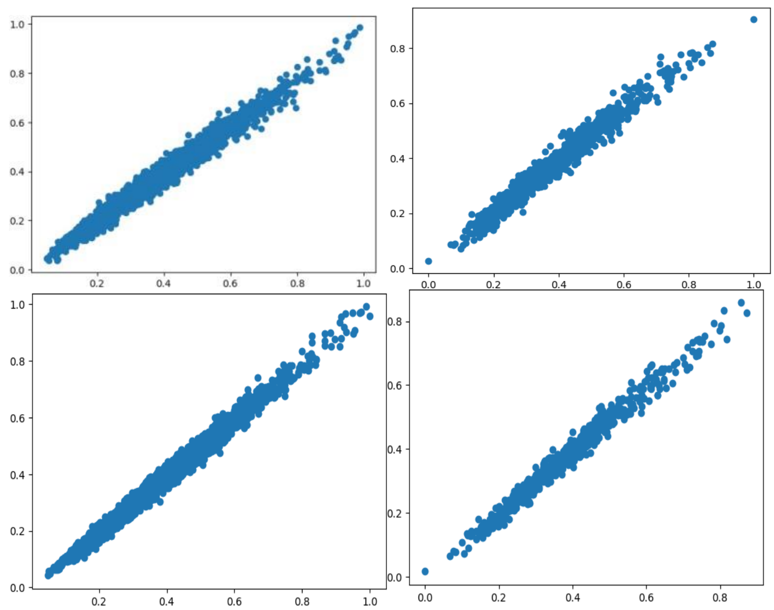


Figure 3: Predicted data Vs True data (both Training and Testing) for various epoch values as given in the table

II. *Effect of Sample Size of Training*

To under the effect of the sample size of the training dataset, keeping other parameters constant as given in table 1, the size of the training data was varied. It was observed that when the training data is 70% of the total data, the model gave the best performance in predicting the output. It was interesting to observe that when the training data was 80% of the total data, the prediction results were inappropriate (negative value of R^2). This justifies that enough data has to be provided for validating and testing the model. Hence 70% for Training, 15% for Validation and 15% for Testing gave a better performance than the other sizes of training data. This was fixed for further analysis

SIZE OF TRAINING DATA	SIZE OF VALIDATION DATA	SIZE OF TESTING DATA	TRAINING		**TESTING**	
			R^2	MSE	$\mathbf{R^2}$	**MSE**
50%	25%	25%	0.975	0.00061	0.962	0.00089
60%	20%	20%	0.977	0.00055	0.964	0.0008
70%	15%	15%	0.99757120944125 65	5.14099487754052 95e-05	0.9963028957637 146	8.683820129706703 e-05
80%	10%	10%	Inappropriate Result.			



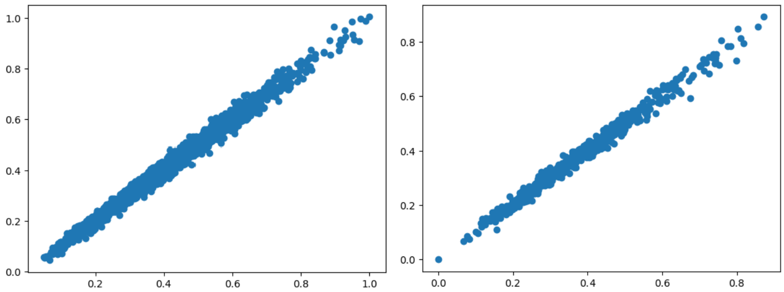


Figure 5: Predicted data Vs True data (Both Training and Testing) for various training data size

III. Effect of the number of Hidden Layer

The number of hidden layers of an ANN model plays an important role in the architecture of the neural network. To find the optimal number of hidden layers, additional layers were added to the architecture given in **table 1.** Each additional layer contained 100 nodes and RELU as the activation function. From the analysis, it was evident that when the number of hidden layers was increased, the performance of the ANN model decreased.

NUMBER OF HIDDEN LAYERS	**TRAINING**		**TESTING**	
	R^2	**MSE**	**R^2**	**MSE**
2	0.9958217015670963	8.844159395739491e-05	0.9964443723902479	8.351517468257392e-05
3	0.9846823154953979	0.000324227781973441 63	0.9843752833623037	0.0003669959519337558
4	0.9975712094412565	5.14099487754052 95e-05	0.9963028957637146	8.683820129706703e-05

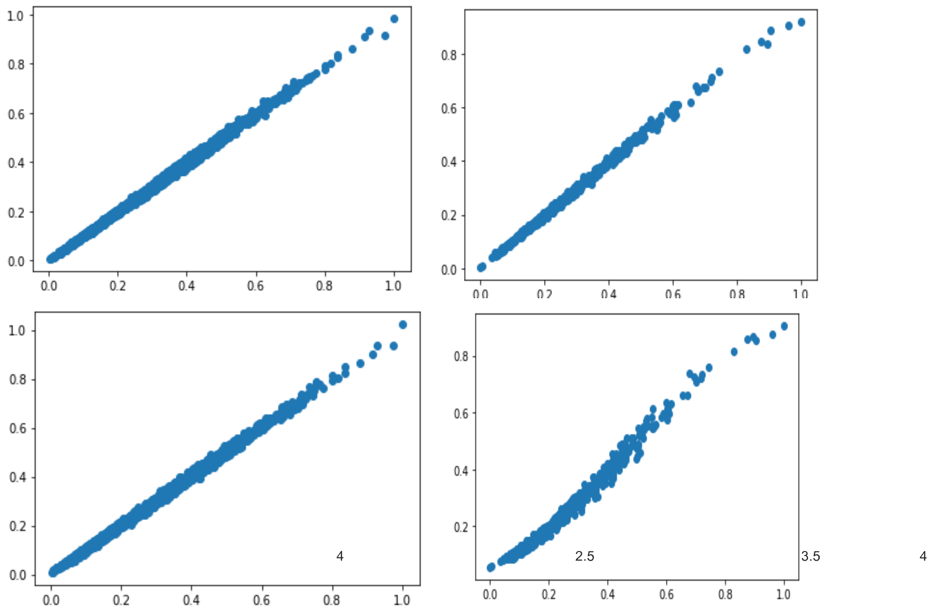
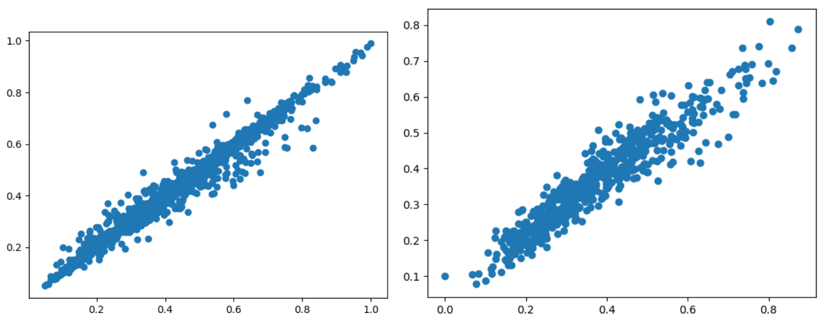


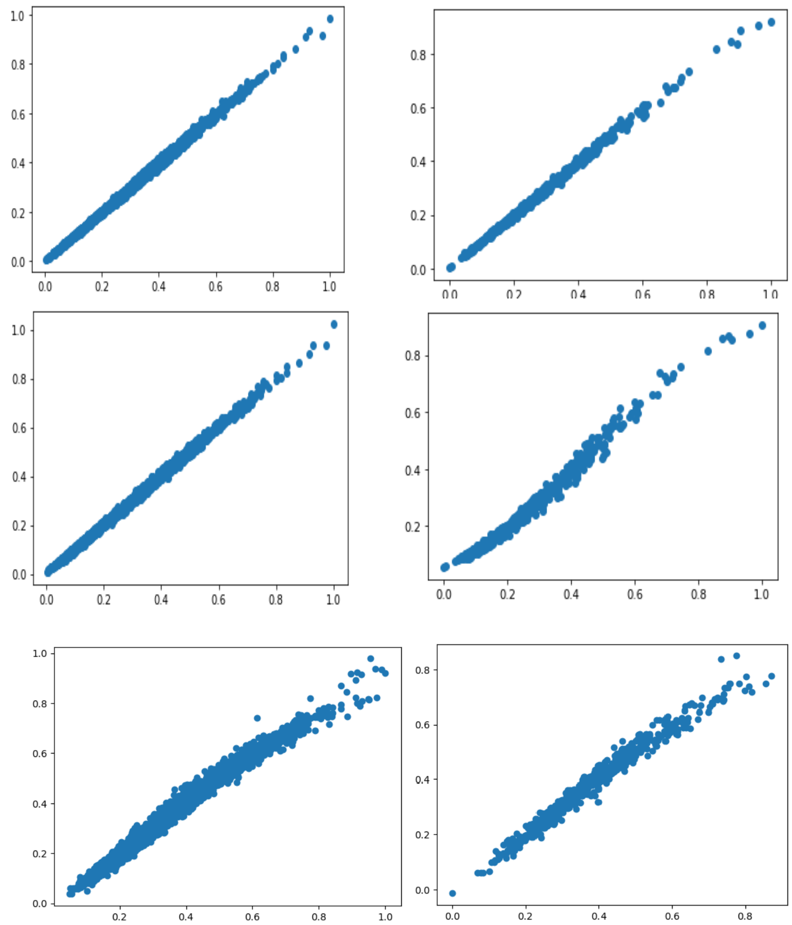
figure 7: Predicted data Vs True data (Both Training and Testing) for variation in the number of the hidden layer.

IV. *Effect of Activation Function*

Once the number of the hidden layer is fixed, the combination of the activation function in the hidden layer has been decided. Initially, the combination of activation functions was started with RELU in all the layers. Then LINEAR activation function was introduced in the first hidden layer. It was observed that, once the LINEAR activation function was introduced in the first hidden layer, the performance of the model increased. Hence the two hidden layers were fixed with the LINEAR-RELU combination. The activation function in the output layer was varied again. TANH and SIGMOID activation functions were used. It was observed that the combination LINEAR – RELU -RELU gave the best prediction results compared to the other combinations. During all the trials, the first hidden layer with LINEAR activation function and the second hidden layer with RELU gave the best performance.



COMBINATION	ACTIVATION FUNCTION	TRAINING		TESTING	
		R^2	MSE	R^2	MSE
1	RELU-RELU-RELU	0.979	0.00049	0.895	0.0025
2	**LINEAR-RELU-RELU**	0.9975712094412565	5.1409948775405295e-05	0.996302895763 7146	8.683820129706703e-05
3	LINEAR-RELU-TANH	0.9886	0.00027	0.987	0.0006
4	LINEAR-RELU-SIGMOID	0.9846823154953979	0.00032422778197344163	0.9843752833623037	0.0003669959519337558



V. *Effect of Number of Nodes*

The number of Nodes in the hidden layers of the neural network is the important parameter. The number of nodes plays an important role in the performance and predicting capability of the model. The number of nodes in the two hidden layers was varied and the topology which gave the best prediction was found. It was observed that topology 7-90-100-1 gave the best prediction and performance.

COMBINATION	NUMBER OF NODES	TRAINING		TESTING	
		R^2	MSE	R^2	MSE
1	7-40-50-1	0.586	0.01	0.477	0.012
2	**7-90-100-1**	**0.987**	**0.00037**	**0.976**	**0.00032**
3	1-140-150-1	0.98	0.00047	0.962	0.0009

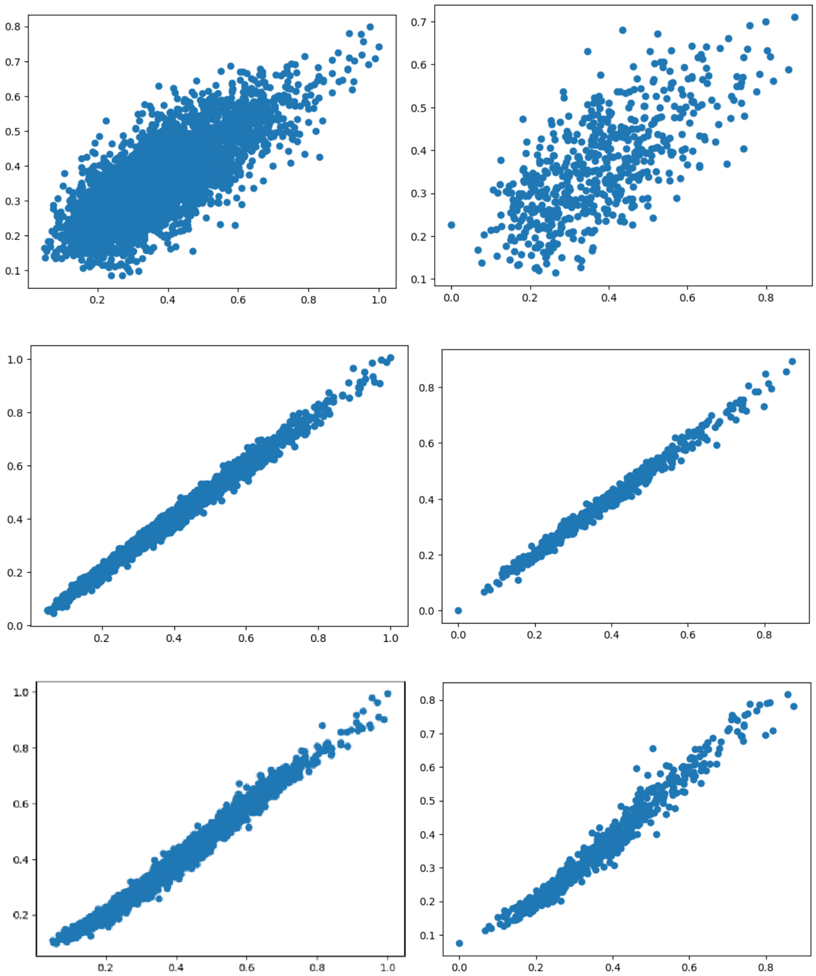


Figure 11: Predicted data Vs True data (Both Training and Testing) for variation of nodes in hidden layer.

5. CONCLUSION

An attempt was made to develop an ANN model for the styblinski-tang function with n=3. The data for ANN modelling was successfully generated using the function. The network with two hidden layers, LINEAR – RELU – RELU activation combination with 3 – 90 -100-1 topology gave the best performance among all the networks. When comparing adam and RMSProp optimizers, the former gave a better performance by reducing the loss function than the latter. Each parameter was tuned in such a way that the neural network gave the best prediction. Based on the sensitivity analysis, when the epoch was 750 and the training data size was 70% of the total generated data, the model performed better compared to the other trials. Based on the complexity and randomness of the input data, the neural network couldn’t give a poor performance at lower node values. Hence to improve the performance, a higher number of nodes was used. Furthermore, the node can be increased, but it can increase the number of tuneable weight and biases which is undesirable.

6. REFERENCE

1) https://www.sfu.ca/~ssurjano/stybtang.html
2) https://www.analyticsvidhya.com/blog/2021/08/a-walk-through-of-regression-analysis-using-artificial-neural-networks-in-tensorflow/
3) https://towardsdatascience.com/keras-101-a-simple-and-interpretable-neural-network-model-for-house-pricing-regression-31b1a77f05ae
4) https://www.tensorflow.org/tutorials/keras/regression
5) https://matplotlib.org/stable/index.html

ANNEXURE – I

1. CODE FOR DATA GENERATION, NORMALIZATION AND ANN MODELLING
2. CODE FOR THE EFFECT OF EPOCH
3. CODE FOR THE EFFECT OF SIZE OF TRAINING DATA
4. CODE FOR THE EFFECT OF THE NUMBER OF HIDDEN LAYERS
5. CODE FOR THE EFFECT OF ACTIVATION FUNCTION
6. CODE FOR THE EFFECT OF NUMBER OF NODES

In [1]:

```python

import numpy as np
import pandas as pd
import random

dataset = 4000

variables = 3

X = np.random.uniform(-5.0, 5.0, size = (dataset, variables))   # Generating Random Data uniformly within the range
```

In [2]:

```python
ST = []
for i in range (dataset):
    total = 0
    for j in range(variables):
        total = total+((X[i][j])**4 - 16*(X[i][j])**2+5*(X[i][j]))  # Test Function Calculation
        Total=total/2
    ST.append(Total)
```

In [3]:

```python
data = pd.concat([pd.DataFrame(X),pd.DataFrame(ST)], axis=1)       # Concatenation of input and output data

```

In [5]:

```python
.to_excel('C:\\Users\\asus\\OneDrive\\Documents\\Anvesh_project\\datageneration.xlsx')   #Writing the Data to Excel File
```

Normalization of data

In [6]:

```python
Data=pd.read_excel('C:\\Users\\asus\\OneDrive\\Documents\\Anvesh_project\\datageneration.xlsx')
```

In [7]:

```python
data1=Data.iloc[:,1:9]
from sklearn.preprocessing import MinMaxScaler      # To preprocess the data and Normalise the data
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data1.values)
```

In [8]:

```python
scaled_data.shape
```

Out[8]: (4000, 4)

In [9]:

```python
x = scaled_data[:,0:3]
y = scaled_data[:,3]
```

ANN modeling Adam as optimizer

In [11]:

```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.15, random_state=0)
```

In [12]:

```python
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

In [13]:

```python
net = Sequential()
net.add(Dense(90,input_dim=3,activation='linear'))        # Linear Activation Function
net.add(Dense(100,activation='relu'))                     # Rectified Linear Unit Activation Function
net.add(Dense(1,activation='relu'))                       # Rectified Linear Unit Activation Function
```

In [14]:

```python
net.compile(optimizer = 'adam', loss = 'MSE')
print(net.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 90)                360       
                                                                 
 dense_1 (Dense)             (None, 100)               9100      
                                                                 
 dense_2 (Dense)             (None, 1)                 101       
                                                                 
=================================================================
Total params: 9,561
Trainable params: 9,561
Non-trainable params: 0
_________________________________________________________________
None
```

In [15]:
```
history = net.fit(x_train, y_train, epochs = 500, validation_split=0.15)
```
```
91/91 [==============================] - 0s 3ms/step - loss: 0.0084 - val_loss: 0.0090
Epoch 15/500
91/91 [==============================] - 0s 3ms/step - loss: 0.0083 - val_loss: 0.0082
Epoch 16/500
91/91 [==============================] - 0s 3ms/step - loss: 0.0077 - val_loss: 0.0075
Epoch 17/500
91/91 [==============================] - 0s 5ms/step - loss: 0.0074 - val_loss: 0.0070
Epoch 18/500
91/91 [==============================] - 0s 3ms/step - loss: 0.0070 - val_loss: 0.0072
Epoch 19/500
91/91 [==============================] - 0s 3ms/step - loss: 0.0064 - val_loss: 0.0061
Epoch 20/500
91/91 [==============================] - 0s 3ms/step - loss: 0.0057 - val_loss: 0.0056
Epoch 21/500
91/91 [==============================] - 0s 3ms/step - loss: 0.0053 - val_loss: 0.0061
Epoch 22/500
91/91 [==============================] - 0s 5ms/step - loss: 0.0046 - val_loss: 0.0046
Epoch 23/500
91/91 [==============================] - 0s 3ms/step - loss: 0.0040 - val_loss: 0.0049
```

In [16]:
```
from sklearn.metrics import r2_score
```

In [17]:
```
y_train_pre = net.predict(x_train)
```
```
107/107 [==============================] - 0s 2ms/step
```

In [18]:
```
r2_train = r2_score(y_train, y_train_pre)
r2_train
```

Out[18]:
```
0.9971310667387485
```

In [19]:
```
y_test_pre=net.predict(x_test)
r2_test = r2_score(y_test, y_test_pre)
r2_test
```
```
19/19 [==============================] - 0s 2ms/step
```

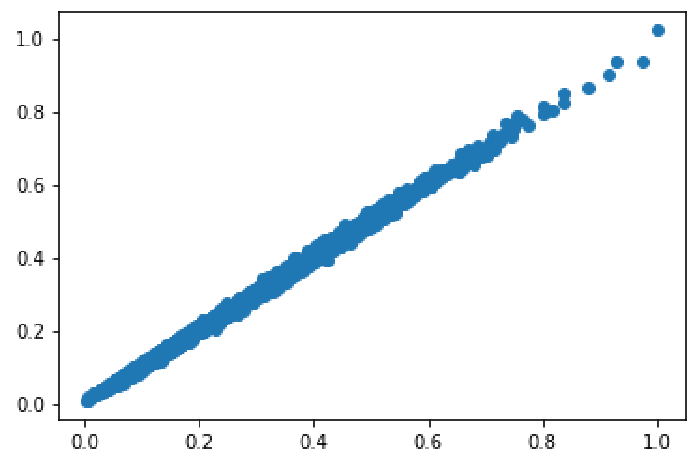
Out[19]:
```
0.9974495026359527
```

Scatter plot

In [20]:
```
from matplotlib import pyplot as p
```

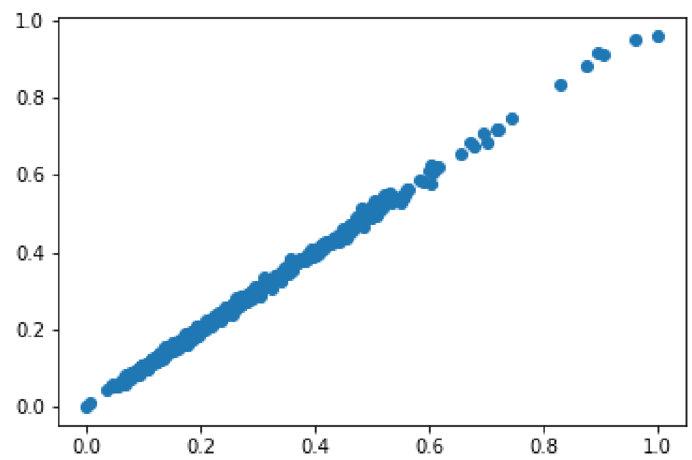
In [21]:
```
p.scatter(y_train, y_train_pre)
```

Out[21]:
```
<matplotlib.collections.PathCollection at 0x19df944adc0>
```



In [22]:
```
p.scatter(y_test, y_test_pre)
```

Out[22]:
```
<matplotlib.collections.PathCollection at 0x19dfa512a00>
```



In [23]:
```python
from sklearn.metrics import mean_squared_error
mse_train=mean_squared_error(y_train,y_train_pre)
mse_train
```
Out[23]: 6.0726402064615814e-05

In [24]:
```python
mse_test=mean_squared_error(y_test, y_test_pre)
mse_test
```
Out[24]: 5.990650772922385e-05

ANN modeling RMS as optimizer

In [27]:
```python
net = Sequential()
net.add(Dense(90,input_dim=3,activation='linear'))
net.add(Dense(100,activation='relu'))
net.add(Dense(1,activation='relu'))
```

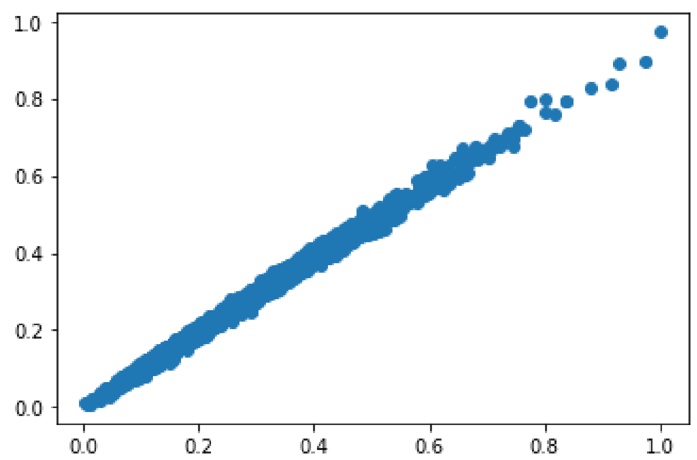
In [28]:
```python
net.compile(optimizer = 'RMSProp', loss = 'MSE')
history = net.fit(x_train, y_train, epochs = 500, validation_split=0.15)
```
```
Epoch 383/500
91/91 [==============================] - 0s 3ms/step - loss: 3.8764e-04 - val_loss: 0.0012

Epoch 384/500
91/91 [==============================] - 0s 3ms/step - loss: 3.6640e-04 - val_loss: 5.7752e-04
Epoch 385/500
91/91 [==============================] - 0s 3ms/step - loss: 4.1711e-04 - val_loss: 0.0019
Epoch 386/500
91/91 [==============================] - 0s 4ms/step - loss: 3.7522e-04 - val_loss: 0.0012
Epoch 387/500
91/91 [==============================] - 0s 3ms/step - loss: 3.9538e-04 - val_loss: 9.7641e-04
Epoch 388/500
91/91 [==============================] - 0s 3ms/step - loss: 3.8612e-04 - val_loss: 4.5683e-04
Epoch 389/500
91/91 [==============================] - 0s 3ms/step - loss: 3.7172e-04 - val_loss: 3.5779e-04
Epoch 390/500
91/91 [==============================] - 0s 3ms/step - loss: 3.8571e-04 - val_loss: 1.3493e-04
Epoch 391/500
91/91 [==============================] - 0s 3ms/step - loss: 3.6124e-04 - val_loss: 5.6736e-04
Epoch 392/500
```

In [29]:
```python
y_train_pre = net.predict(x_train)
```

In [30]:
```python
r2_train = r2_score(y_train, y_train_pre)
r2_train
```
Out[30]: 0.9922120717480983

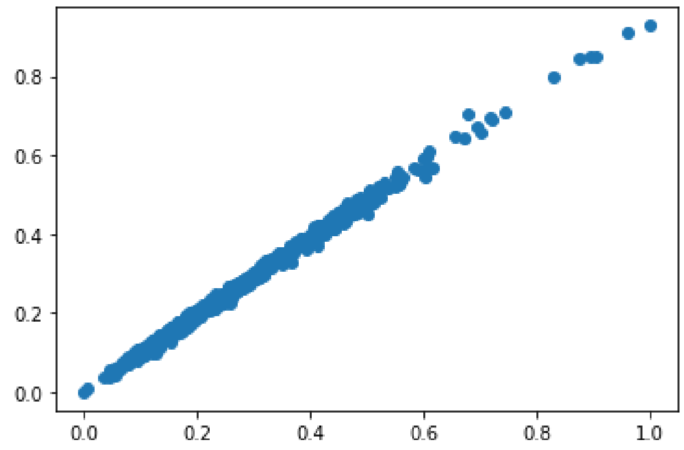
In [31]:
```python
y_test_pre=net.predict(x_test)
r2_test = r2_score(y_test, y_test_pre)
r2_test
```
```
19/19 [==============================] - 0s 2ms/step
```
Out[31]: 0.9933728901679092

In [32]:
```python
p.scatter(y_train, y_train_pre)
```
Out[32]: <matplotlib.collections.PathCollection at 0x19df7150040>



In [33]:
```python
p.scatter(y_test, y_test_pre)
```
Out[33]: <matplotlib.collections.PathCollection at 0x19df711dcd0>

In [33]:
```
p.scatter(y_test, y_test_pre)
```
Out[33]: <matplotlib.collections.PathCollection at 0x19df711dcd0>



In [34]:
```
mse_tr=mean_squared_error(y_train,y_train_pre)
mse_tr
```
Out[34]: 0.00016484624046955119

In [35]:
```
mse_te=mean_squared_error(y_test, y_test_pre)
mse_te
```
Out[35]: 0.0001556586617084596

In [ ]:

#Effect of activation Function

In [1]:
```
import numpy as np
import pandas as pd
```
In [2]:
```
Data=pd.read_excel('C:\\Users\\asus\\OneDrive\\Documents\\Anvesh_project\\datageneration.xlsx')
```
In [3]:
```
data1=Data.iloc[:,1:5]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data1.values)
```
In [4]:
```
x = scaled_data[:,0:3]
y = scaled_data[:,3]
```
In [5]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.15, random_state=0)
```
In [6]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```
In [7]:
```
net = Sequential()
net.add(Dense(90,input_dim=3,activation='linear'))
net.add(Dense(100,activation='relu'))
net.add(Dense(1,activation='sigmoid'))
```
EW22M

In [8]:
```
net.compile(optimizer = 'adam', loss = 'MSE')
print(net.summary())
```
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 90)                360       
                                                                 
 dense_1 (Dense)             (None, 100)               9100      
                                                                 
 dense_2 (Dense)             (None, 1)                 101       
                                                                 
=================================================================
Total params: 9,561
Trainable params: 9,561
Non-trainable params: 0
```

In [9]:
```python
history = net.fit(x_train, y_train, epochs = 750, validation_split=0.15)
```

```
91/91 [==============================] - 0s 4ms/step - loss: 0.0044 - val_loss: 0.0047
Epoch 49/750
91/91 [==============================] - 0s 3ms/step - loss: 0.0045 - val_loss: 0.0041
Epoch 50/750
91/91 [==============================] - 0s 3ms/step - loss: 0.0044 - val_loss: 0.0040
Epoch 51/750
91/91 [==============================] - 0s 3ms/step - loss: 0.0041 - val_loss: 0.0038
Epoch 52/750
91/91 [==============================] - 0s 3ms/step - loss: 0.0042 - val_loss: 0.0051
Epoch 53/750
91/91 [==============================] - 0s 3ms/step - loss: 0.0042 - val_loss: 0.0045
Epoch 54/750
91/91 [==============================] - 0s 3ms/step - loss: 0.0040 - val_loss: 0.0035
Epoch 55/750
91/91 [==============================] - 0s 3ms/step - loss: 0.0036 - val_loss: 0.0037
Epoch 56/750
91/91 [==============================] - 0s 3ms/step - loss: 0.0036 - val_loss: 0.0034
Epoch 57/750
91/91 [==============================] - 0s 3ms/step - loss: 0.0032 - val_loss: 0.0031
Epoch 58/750
```

In [10]:
```python
from sklearn.metrics import r2_score
y_train_pre = net.predict(x_train)
r2_train = r2_score(y_train, y_train_pre)
r2_train
```

```
107/107 [==============================] - 1s 3ms/step
```

Out[10]: 0.9805818796079804

In [11]:
```python
y_test_pre=net.predict(x_test)
r2_test = r2_score(y_test, y_test_pre)
r2_test
```

```
19/19 [==============================] - 0s 2ms/step
```

Out[11]: 0.9788422368794105

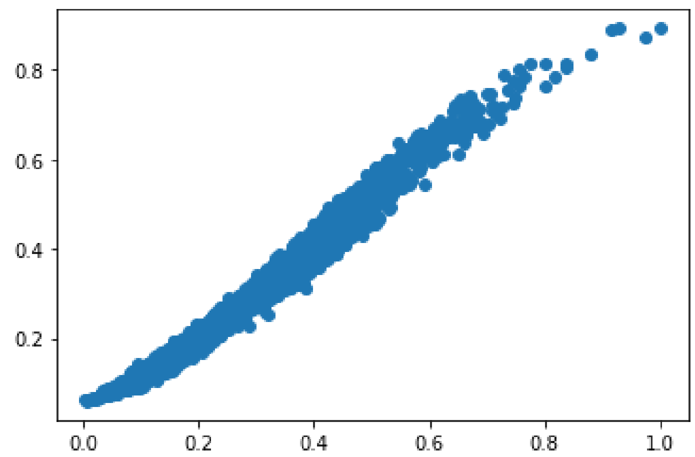
In [12]:
```python
from sklearn.metrics import mean_squared_error
mse_train=mean_squared_error(y_train,y_train_pre)
mse_train
```

Out[12]: 0.0004110212677971129

In [14]:
```python
from matplotlib import pyplot as p
```

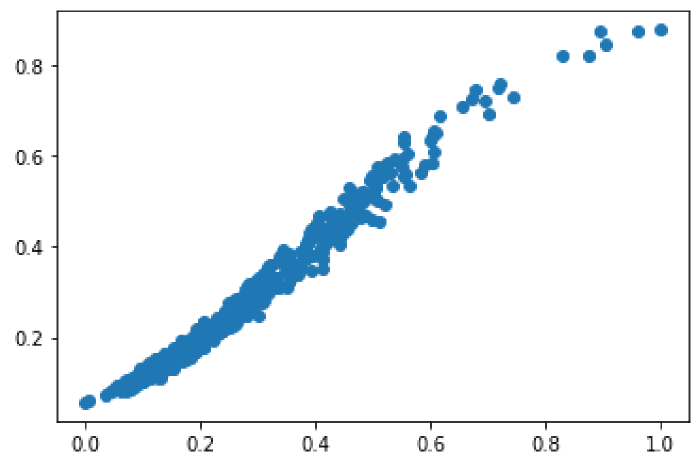
In [15]:
```python
p.scatter(y_train, y_train_pre)
```

Out[15]: <matplotlib.collections.PathCollection at 0x1d6ff6bcfd0>



In [16]:
```python
p.scatter(y_test, y_test_pre)
```

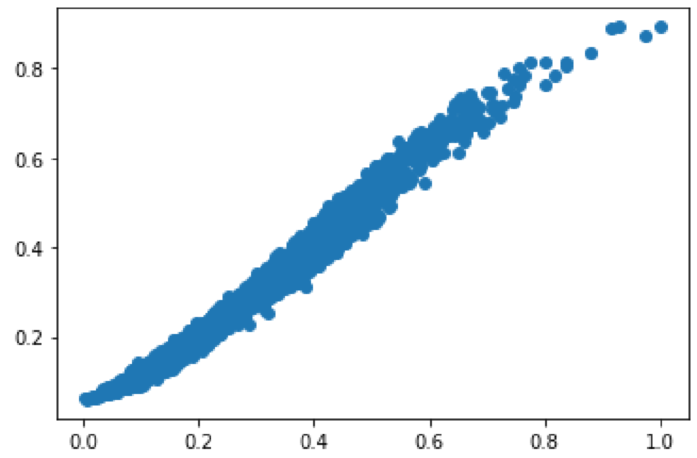
Out[16]: <matplotlib.collections.PathCollection at 0x1d680089e80>



In [14]:
```python
from matplotlib import pyplot as p
```

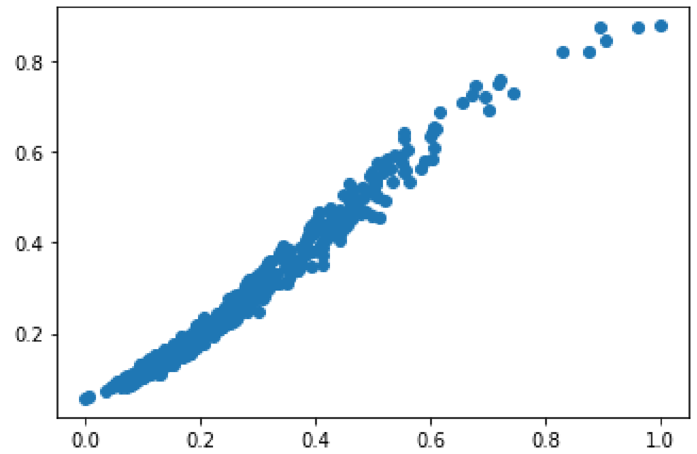
In [15]:
```python
p.scatter(y_train, y_train_pre)
```

Out[15]: <matplotlib.collections.PathCollection at 0x1d6ff6bcfd0>



In [16]:
```python
p.scatter(y_test, y_test_pre)
```

Out[16]: <matplotlib.collections.PathCollection at 0x1d680089e80>



#Effect of Hidden Layers

In [1]:
```python
import numpy as np
import pandas as pd
Data=pd.read_excel('C:\\Users\\asus\\OneDrive\\Documents\\Anvesh_project\\datageneration.xlsx')
```

In [2]:
```python
data1=Data.iloc[:,1:5]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data1.values)
```

In [3]:
```python
x = scaled_data[:,0:3]
y = scaled_data[:,3]
```

In [4]:
```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.15, random_state=0)
```

In [5]:
```python
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

In [7]:
```python
net = Sequential()
net.add(Dense(90,input_dim=3,activation='linear'))
net.add(Dense(100,activation='relu'))
net.add(Dense(90,activation ='relu'))
net.add(Dense(90,activation= 'relu'))
net.add(Dense(1,activation='relu'))
```

In [8]:
```python
net.compile(optimizer = 'adam', loss = 'MSE')
```

In [9]:
```python
print(net.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 90)                360

 dense_6 (Dense)             (None, 100)               9100

 dense_7 (Dense)             (None, 90)                9090

 dense_8 (Dense)             (None, 90)                8190

 dense_9 (Dense)             (None, 1)                 91
```

```
_________________________________________________________________
None
```

In [10]:

```python
history = net.fit(x_train, y_train, epochs = 750, validation_split=0.15)
```

```
91/91 [==============================] - 0s 3ms/step - loss: 3.5689e-04 - val_loss: 0.0011
Epoch 122/750
91/91 [==============================] - 0s 3ms/step - loss: 4.5062e-04 - val_loss: 1.3721e-04
Epoch 123/750
91/91 [==============================] - 0s 3ms/step - loss: 6.0281e-04 - val_loss: 8.4015e-04
Epoch 124/750
91/91 [==============================] - 0s 3ms/step - loss: 3.3651e-04 - val_loss: 1.9677e-04
Epoch 125/750
91/91 [==============================] - 0s 3ms/step - loss: 2.6671e-04 - val_loss: 1.7632e-04
Epoch 126/750
91/91 [==============================] - 0s 3ms/step - loss: 1.5589e-04 - val_loss: 1.0249e-04
Epoch 127/750
91/91 [==============================] - 0s 3ms/step - loss: 1.4933e-04 - val_loss: 3.0348e-04
Epoch 128/750
91/91 [==============================] - 0s 3ms/step - loss: 1.4855e-04 - val_loss: 2.4333e-04
Epoch 129/750
91/91 [==============================] - 1s 6ms/step - loss: 2.0373e-04 - val_loss: 3.8454e-04
Epoch 130/750
91/91 [==============================] - 0s 4ms/step - loss: 4.6062e-04 - val_loss: 3.7386e-04
```

In [11]:

```python
from sklearn.metrics import r2_score
y_train_pre = net.predict(x_train)
r2_train = r2_score(y_train, y_train_pre)
r2_train
```

```
107/107 [==============================] - 2s 4ms/step
```

Out[11]: 0.9975712094412565

In [12]:

```python
y_test_pre=net.predict(x_test)
r2_test = r2_score(y_test, y_test_pre)
r2_test
```

```
19/19 [==============================] - 0s 2ms/step
```

Out[12]: 0.9963028957637146

In [13]:

```python
from sklearn.metrics import mean_squared_error
mse_train=mean_squared_error(y_train,y_train_pre)
mse_train
```

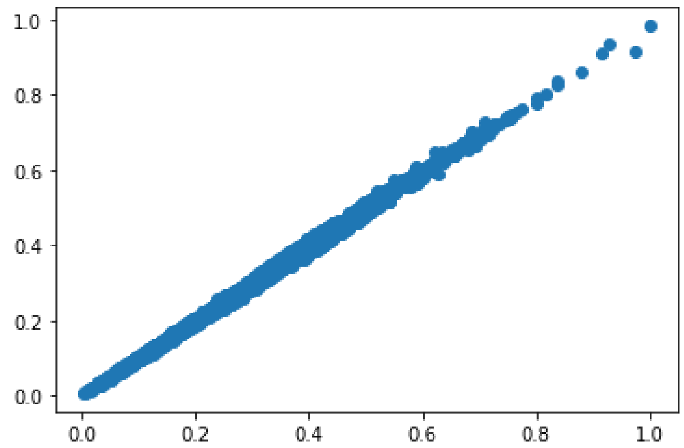
In [15]:

```python
from matplotlib import pyplot as p
```

In [16]:

```python
p.scatter(y_train, y_train_pre)
```

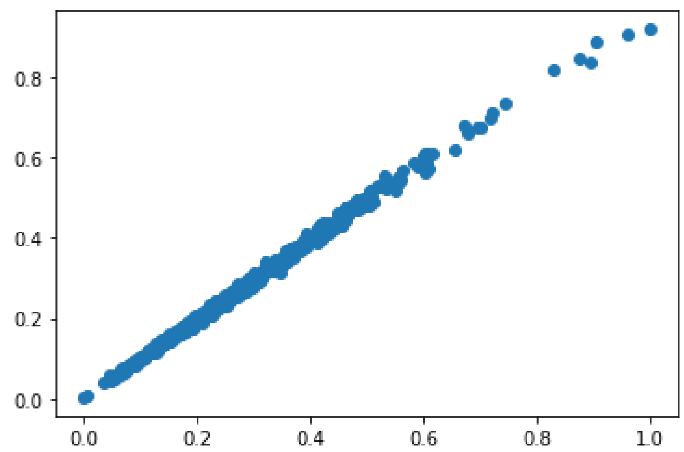
Out[16]: <matplotlib.collections.PathCollection at 0x245dae04f70>



In [17]:

```python
p.scatter(y_test, y_test_pre)
```

Out[17]: <matplotlib.collections.PathCollection at 0x245daef3af0>



In [ ]:

#Epoch

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
Data=pd.read_excel('C:\\Users\\asus\\OneDrive\\Documents\\Anvesh_project\\datageneration.xlsx')
```

In [3]:
```
data1=Data.iloc[:,1:5]
from sklearn.preprocessing import MinMaxScaler      # To preprocess the data and Normalise the data
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data1.values)
```

In [4]:
```
x = scaled_data[:,0:3]
y = scaled_data[:,3]
```

In [5]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.15, random_state=0)
```

In [6]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

In [7]:
```
net = Sequential()
net.add(Dense(90,input_dim=3,activation='linear'))
net.add(Dense(100,activation='relu'))
net.add(Dense(1,activation='relu'))
```

In [8]:
```
net.compile(optimizer = 'adam', loss = 'MSE')
```

In [9]:
```
print(net.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 90)                360

 dense_1 (Dense)             (None, 100)               9100

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 9,561
Trainable params: 9,561
```

In [10]:
```
history = net.fit(x_train, y_train, epochs = 1000, validation_split=0.15)
```

```
91/91 [==============================] - 0s 4ms/step - loss: 8.1929e-05 - val_loss: 7.3344e-05
Epoch 476/1000
91/91 [==============================] - 0s 5ms/step - loss: 6.8653e-05 - val_loss: 3.8771e-05
Epoch 477/1000
91/91 [==============================] - 0s 4ms/step - loss: 4.4975e-05 - val_loss: 1.0796e-04
Epoch 478/1000
91/91 [==============================] - 0s 4ms/step - loss: 1.0378e-04 - val_loss: 2.2137e-04
Epoch 479/1000
91/91 [==============================] - 0s 4ms/step - loss: 3.8641e-04 - val_loss: 2.5020e-04
Epoch 480/1000
91/91 [==============================] - 0s 4ms/step - loss: 1.5827e-04 - val_loss: 2.6139e-04
Epoch 481/1000
91/91 [==============================] - 0s 4ms/step - loss: 2.1083e-04 - val_loss: 1.3333e-04
Epoch 482/1000
91/91 [==============================] - 0s 4ms/step - loss: 6.8201e-05 - val_loss: 3.3127e-05
Epoch 483/1000
91/91 [==============================] - 0s 4ms/step - loss: 4.8992e-05 - val_loss: 4.6436e-05
Epoch 484/1000
91/91 [==============================] - 0s 4ms/step - loss: 1.0119e-04 - val_loss: 8.4682e-05
Epoch 485/1000
```

In [11]:
```
from sklearn.metrics import r2_score
```

In [12]:
```
y_train_pre = net.predict(x_train)
```

```
107/107 [==============================] - 1s 2ms/step
```

In [13]:
```
r2_train = r2_score(y_train, y_train_pre)
r2_train
```

Out[13]:
```
0.9958217015670963
```

In [14]:
```
y_test_pre=net.predict(x_test)
r2_test = r2_score(y_test, y_test_pre)
r2_test
```

```
19/19 [==============================] - 0s 2ms/step
```

Out[14]:
```
0.9964443723902479
```

In [15]:
```
from sklearn.metrics import mean_squared_error
mse_train=mean_squared_error(y_train,y_train_pre)
mse_train
```

Out[15]:
```
8.844159395739491e-05
```

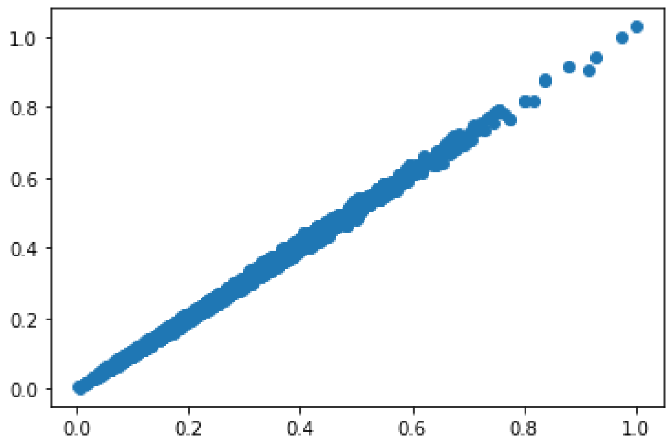
In [16]:
```
mse_test=mean_squared_error(y_test, y_test_pre)
mse_test
```

Out[16]:
```
8.351517468257392e-05
```

In [17]:
```
from matplotlib import pyplot as p
```

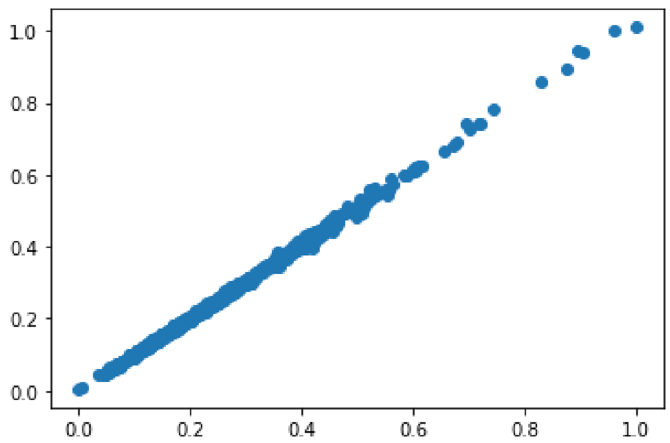
In [18]:
```
p.scatter(y_train, y_train_pre)
```

Out[18]:
```
<matplotlib.collections.PathCollection at 0x21c54254b20>
```



In [19]:
```
p.scatter(y_test, y_test_pre)
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x21c54351190>
```



#Effect of Nodes

In [1]:
```
import numpy as np
import pandas as pd
Data=pd.read_excel('C:\\Users\\asus\\OneDrive\\Documents\\Anvesh_project\\datageneration.xlsx')
```

In [2]:
```
data1=Data.iloc[:,1:5]
from sklearn.preprocessing import MinMaxScaler      # To preprocess the data and Normalise the data
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data1.values)
```

In [3]:
```
x = scaled_data[:,0:3]
y = scaled_data[:,3]
```

In [4]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.15, random_state=0)
```

In [5]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

In [6]:
```
net = Sequential()
net.add(Dense(140,input_dim=3,activation='linear'))
net.add(Dense(150,activation='relu'))
net.add(Dense(1,activation='sigmoid'))
```

In [7]:
```
net.compile(optimizer = 'adam', loss = 'MSE')
```

In [8]:
```
history = net.fit(x_train, y_train, epochs = 200, validation_split=0.15)
```

```
Epoch 102/200
91/91 [==============================] - 1s 6ms/step - loss: 4.8240e-04 - val_loss: 5.0506e-04
Epoch 103/200
91/91 [==============================] - 0s 5ms/step - loss: 4.2839e-04 - val_loss: 4.5733e-04
Epoch 104/200
91/91 [==============================] - 0s 5ms/step - loss: 5.0133e-04 - val_loss: 7.0792e-04
Epoch 105/200
91/91 [==============================] - 0s 5ms/step - loss: 4.6631e-04 - val_loss: 5.8599e-04
Epoch 106/200
91/91 [==============================] - 0s 5ms/step - loss: 5.3359e-04 - val_loss: 5.9028e-04
Epoch 107/200
91/91 [==============================] - 0s 5ms/step - loss: 5.3658e-04 - val_loss: 7.4126e-04
Epoch 108/200
91/91 [==============================] - 0s 4ms/step - loss: 4.7681e-04 - val_loss: 5.2420e-04
```

In [9]:
```
from sklearn.metrics import r2_score
y_train_pre = net.predict(x_train)
r2_train = r2_score(y_train, y_train_pre)
r2_train
```

```
107/107 [==============================] - 1s 3ms/step
```

Out[9]: 0.9846823154953979

In [10]:
```
y_test_pre=net.predict(x_test)
r2_test = r2_score(y_test, y_test_pre)
r2_test
```

```
19/19 [==============================] - 0s 5ms/step
```

Out[10]: 0.9843752833623037

In [11]:
```
from sklearn.metrics import mean_squared_error
mse_train=mean_squared_error(y_train,y_train_pre)
mse_train
```

Out[11]: 0.00032422778197344163

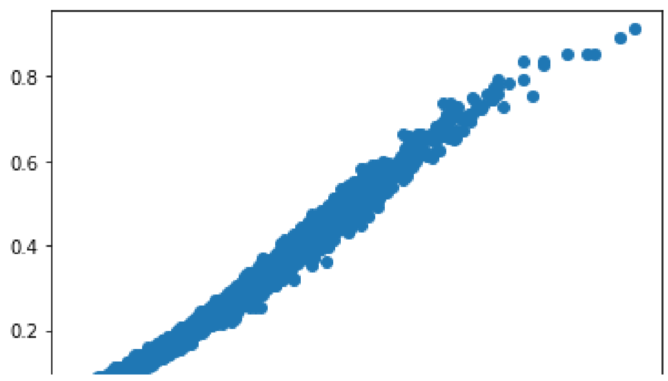
In [12]:
```
mse_test=mean_squared_error(y_test, y_test_pre)
mse_test
```

Out[12]: 0.0003669959519337558

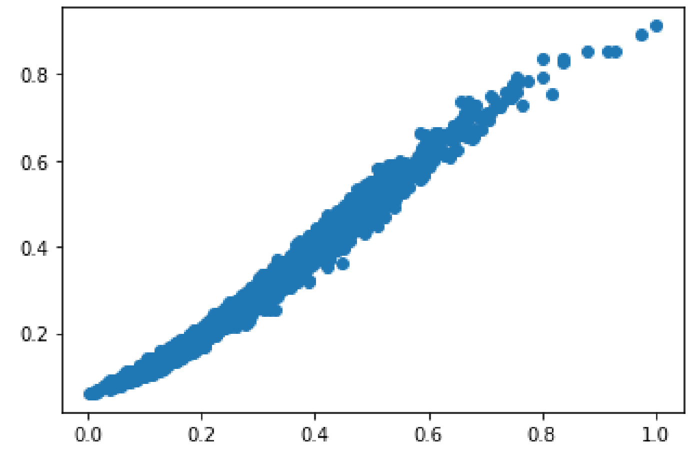
In [13]:
```
from matplotlib import pyplot as p
```

In [14]:
```
p.scatter(y_train, y_train_pre)
```

Out[14]: <matplotlib.collections.PathCollection at 0x211e89469a0>

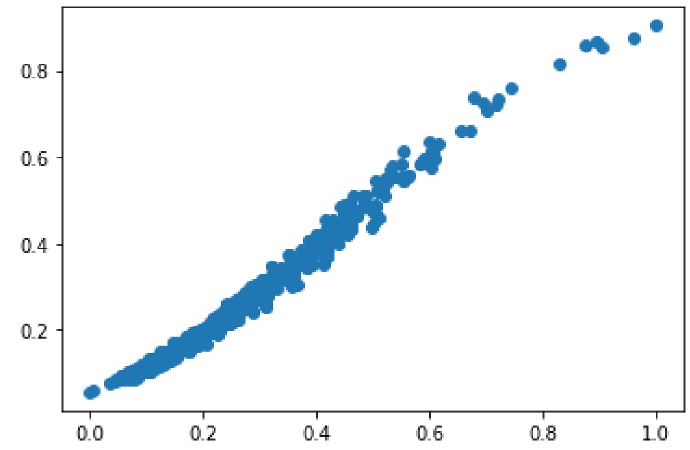


Out[14]: <matplotlib.collections.PathCollection at 0x211e89469a0>



In [15]:
```
p.scatter(y_test, y_test_pre)
```

Out[15]: <matplotlib.collections.PathCollection at 0x211e90d4670>



In [ ]:
```

```