EECS2001 Test 5 - Machine Learning

December 2, 2021 6:15 PM

Professor Jeff Edmonds

Adnan Fahad Faizi
217905928

Shahnaz Ismail
216798712

Function 1:

Let the AbsoluteValue function be $|x^2 - 1|$ that would help the retail store to decide which strawberries to buy at what time of the year.

```
# Function 2: |(x^degree) - 1|
if AP["function"]=="AbsoluteValue":
  y = 1
  for i in range(AP["nDimensions"]):
    if x[i] != 0:
      y *= abs((x[i]**AP["degree"]) - 1)
  return y
```

We change the parameter maxX from 100 to 10 because of two reasons:

1. For slight break in AbsoluteValue function to be more clear.
2. So each floating number from 0.0 - 1.1 to represent a month of the year.

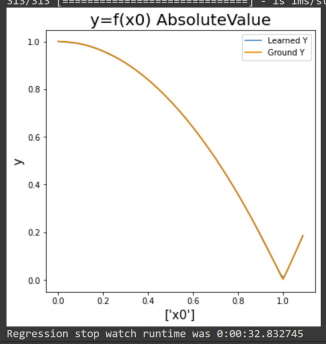
For example: 0.0 represents January
0.1 represents February
.
.
.
1.1 represents December

For the first try we set the value of the parameter "nDimension" = 1. That is, x = [x0]

- Y(x=[x0]) which is the output representing the sweetness of strawberries given the input which is the number of supplies in x0 at different times of the year.

```
sw = StopWatch('Regression')
# Graph 1 # Report on running time
all(["function","AbsoluteValue"], ["maxX", 1.1]) # We have made the maxX parameter 1.1 so the outcome graph looks more clear.
                                                 # and data to be represented in a whole year. (0.0 being January and similarly 1.1 being December)
sw.end()
```

```
Regression stop watch start time: 2021-12-02 20:33:20.759844
Epoch 1/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0049 - mae: 0.0305
Epoch 2/5
3000/3000 [==============================] - 6s 2ms/step - loss: 8.5560e-04 - mae: 0.0178
Epoch 3/5
3000/3000 [==============================] - 6s 2ms/step - loss: 7.4410e-04 - mae: 0.0167
Epoch 4/5
3000/3000 [==============================] - 6s 2ms/step - loss: 6.5274e-04 - mae: 0.0162
Epoch 5/5
3000/3000 [==============================] - 6s 2ms/step - loss: 5.8590e-04 - mae: 0.0152
313/313 [==============================] - 1s 1ms/step - loss: 1.2078e-05 - mae: 0.0023
```




```
Regression stop watch runtime was 0:00:32.832745
```

The learning graph does a perfect job when given the input parameters. Giving us valuable data about when to find sweet strawberries around the year for the retail store and the level of accuracy of the leaning graph indicates how accurate it is in predicting the outcoming result.

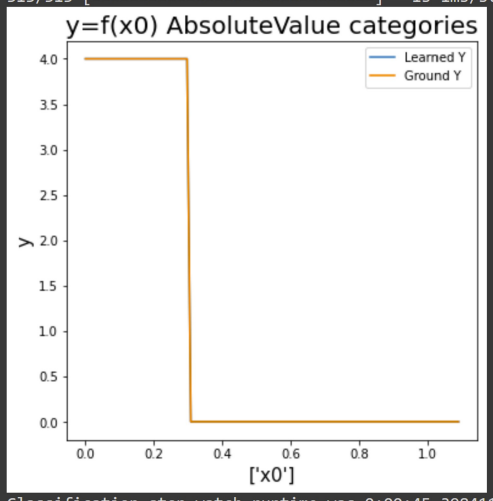
If we categorize our x0 input values into 4 categories of [0.0 (Jan)- 0.3 (April)], [0.3 (April)- 0.6 (Jul)], [0.6 (Jul)- 0.9 (Oct)], [0.9 (Oct) - 1.1 (Dec)], we get the below categorization.

Note that the parameter "categorize" accepts an array of y values given our input values which is the different months of the year. For the catagories to be plotted correctly, we need to do the following calculations:

$$\left|0.3^2 - 1\right| = 0.91$$
$$\left|0.6^2 - 1\right| = 0.64$$
$$\left|0.9^2 - 1\right| = 0.19$$
$$\left|1.1^2 - 1\right| = 0.21$$

```
sw = StopWatch('Classification')
setAP(["function","AbsoluteValue"], ["maxX", 1.1])
all(["categorize",[0.91, 0.64, 0.19, 0.21]])
sw.end()
```

```
Classification stop watch start time: 2021-12-02 23:44:57.516357
Epoch 1/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0274 - accuracy: 0.9945
Epoch 2/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0054 - accuracy: 0.9983
Epoch 3/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0043 - accuracy: 0.9985
Epoch 4/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0037 - accuracy: 0.9986
Epoch 5/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0033 - accuracy: 0.9986
313/313 [==============================] - 1s 1ms/step - loss: 0.0022 - accuracy: 0.9998
```




```
Classification stop watch runtime was 0:00:45.398411
```

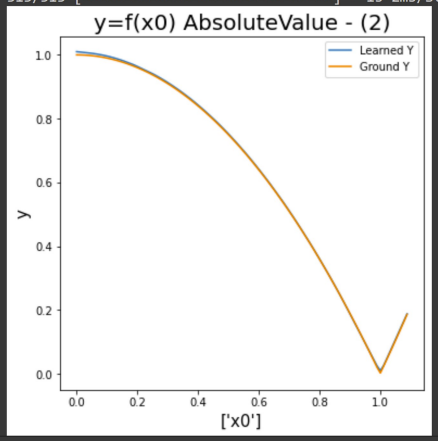
We can interpret the above categorization as its in our best interest to buy strawberries between January - April because according to the testing data, the strawberries are the sweetest at that time. Similarly, according to the learned categorization graph, after the month April, the rest of the year which are the rest of the categories, strawberries are sour and its not in our best interest to buy them for our retail store.

Comparing how the graph learns by trying different degrees of the function AbsoluteValue. In other words, how accurate can the learning graph predict the sweetness of the strawberry in a year.

1. The performance when the parameter "degree" = 2

```
sw = StopWatch('Plotting AP vs loss')
setAP(["maxX", 1.1])
plotAP(True,"degree",[2 , 25, 50], [['function', "AbsoluteValue"], ]) # Comparing degree 2, 25 and 50
sw.end()
```

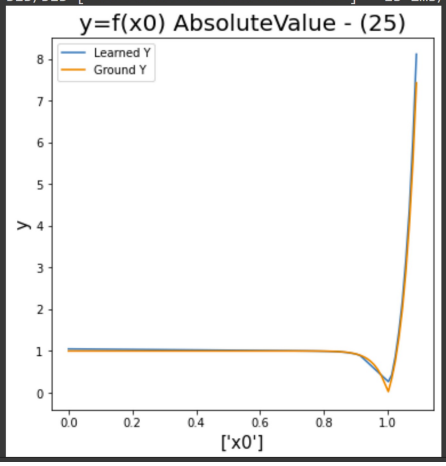
```
Plotting AP vs loss stop watch start time: 2021-12-02 20:35:54.669595
Build for degree = 2
Epoch 1/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0046 - mae: 0.0308
Epoch 2/5
3000/3000 [==============================] - 6s 2ms/step - loss: 8.0825e-04 - mae: 0.0182
Epoch 3/5
3000/3000 [==============================] - 6s 2ms/step - loss: 7.1495e-04 - mae: 0.0167
Epoch 4/5
3000/3000 [==============================] - 6s 2ms/step - loss: 6.2895e-04 - mae: 0.0156
Epoch 5/5
3000/3000 [==============================] - 6s 2ms/step - loss: 5.6607e-04 - mae: 0.0151
313/313 [==============================] - 1s 2ms/step - loss: 1.1577e-04 - mae: 0.0087
```



2. The performance when the parameter "degree" = 25

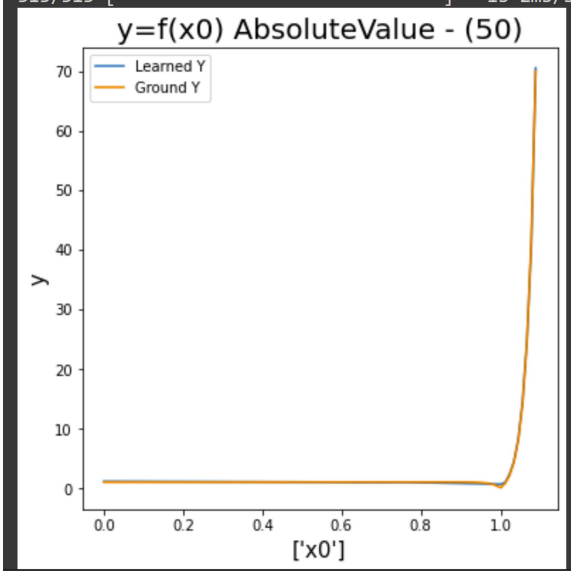
```
sw = StopWatch('Plotting AP vs loss')
setAP(["maxX", 1.1])
plotAP(True,"degree",[2 , 25, 50], [['function', "AbsoluteValue"], ]) # Comparing degree 2, 25 and 50
sw.end()
```

```
['x0']
Build for degree = 25
Epoch 1/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.2262 - mae: 0.1734
Epoch 2/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0164 - mae: 0.0842
Epoch 3/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0125 - mae: 0.0767
Epoch 4/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0112 - mae: 0.0725
Epoch 5/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0104 - mae: 0.0681
313/313 [==============================] - 1s 2ms/step - loss: 0.0111 - mae: 0.0479
```



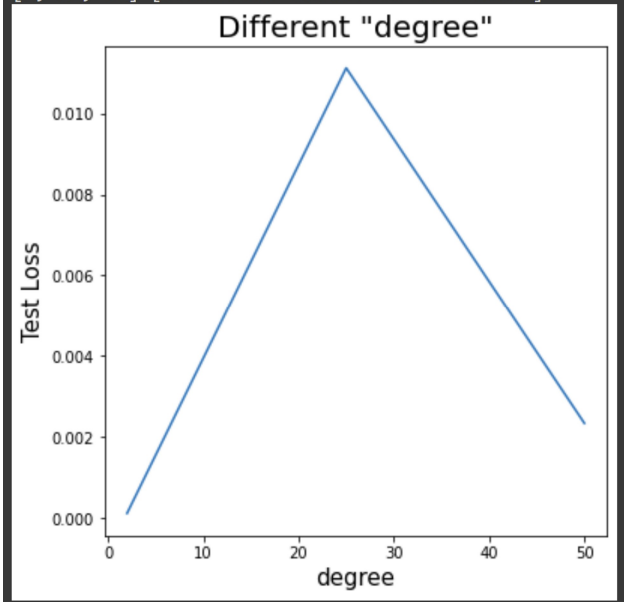
3. The performance when the parameter "degree" = 50

```
Build for degree = 50
Epoch 1/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.2750 - mae: 0.1604
Epoch 2/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0364 - mae: 0.0787
Epoch 3/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0187 - mae: 0.0661
Epoch 4/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0154 - mae: 0.0614
Epoch 5/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0137 - mae: 0.0581
313/313 [==============================] - 1s 2ms/step - loss: 0.0023 - mae: 0.0120
```



The graph with degree 2 is very close to being absolute perfect in learning the sweetness of the strawberry, while increasing its degree to the odd number 25, will increase the Test Loss significantly. Surprisingly, when the degree is increased to 50, the level of accuracy comes marginally closer to the accuracy when the degree is 2.

```
[2, 25, 50] [0.00011577 0.01112085 0.00233911]
```



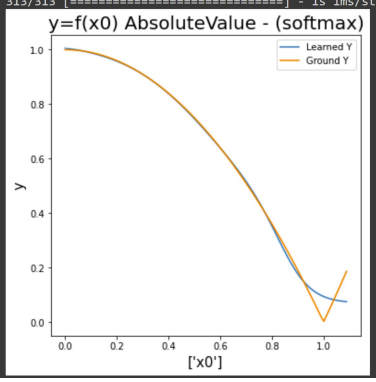

```
Plotting AP vs loss stop watch runtime was 0:02:01.399161
```

Trying different activation to check the performance of learning:

1. For **softmax** activation, the learning function does a good job in predicting the sweetness of strawberry until the month of 0.9 (October), however, it does a poor job afterward.

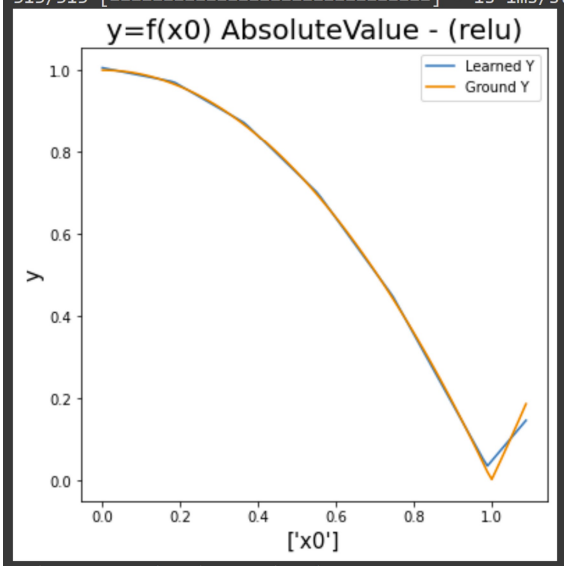
```
sw = StopWatch('Plotting AP vs loss')
plotAP(True,"activation",["softmax", "relu","sigmoid"], [['nHidden', 10], ]) # testing it for the additinal softmax acitivation
sw.end()
```

```
Plotting AP vs loss stop watch start time: 2021-12-02 20:37:56.097732
Build for activation = softmax
Epoch 1/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.1612 - mae: 0.2269
Epoch 2/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0088 - mae: 0.0566
Epoch 3/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0080 - mae: 0.0520
Epoch 4/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0070 - mae: 0.0439
Epoch 5/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0063 - mae: 0.0368
313/313 [==============================] - 1s 1ms/step - loss: 0.0060 - mae: 0.0352
```



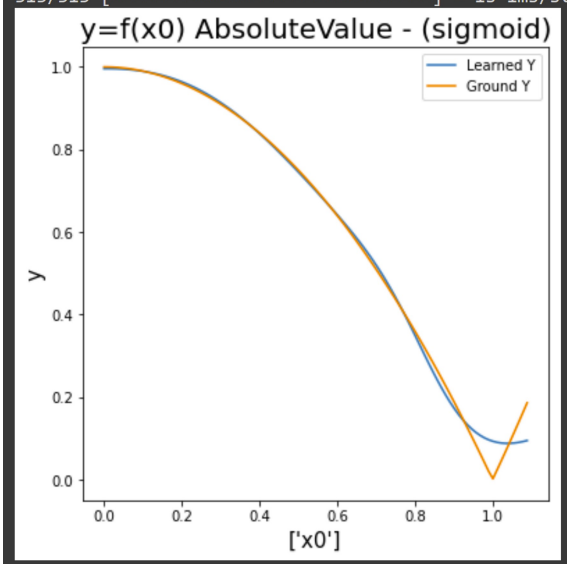
2. For **relu** activation, the learning function does a good job in predicting the sweetness of strawberry until around the month of 1.0 (November), however, it does a poor prediction for the last month of the year.

```
Build for activation = relu
Epoch 1/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0687 - mae: 0.0929
Epoch 2/5
3000/3000 [==============================] - 4s 1ms/step - loss: 0.0054 - mae: 0.0317
Epoch 3/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0033 - mae: 0.0261
Epoch 4/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0020 - mae: 0.0220
Epoch 5/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0012 - mae: 0.0190
313/313 [==============================] - 1s 1ms/step - loss: 7.9428e-04 - mae: 0.0148
```



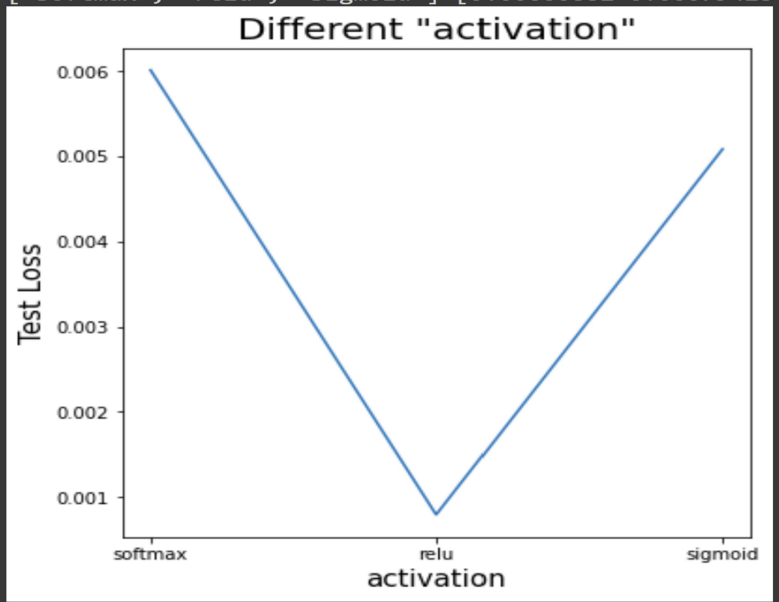
3. For **sigmoid** activation, the learning function does a good job in predicting the sweetness of strawberry until around the month of 0.8 (September) before getting inaccurate.

```
Build for activation = sigmoid
Epoch 1/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.1535 - mae: 0.2063
Epoch 2/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0103 - mae: 0.0587
Epoch 3/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0089 - mae: 0.0522
Epoch 4/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0070 - mae: 0.0417
Epoch 5/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0056 - mae: 0.0393
313/313 [==============================] - 1s 1ms/step - loss: 0.0051 - mae: 0.0371
```



As a comparison, the learning is reported as most accurate in **relu** activiation and the least accurate in **softmax** activation, while doing an okay prediction in **sigmoid** activation.

```
['softmax', 'relu', 'sigmoid'] [0.00600882 0.00079428 0.00508189]
```



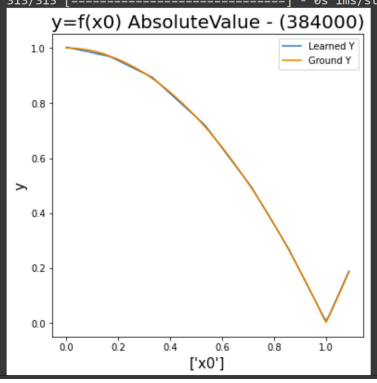

```
Plotting AP vs loss stop watch runtime was 0:02:00.521255
```

Increasing the training data to check the performance of learning:

The learner graph correctly predicates the ground graph with high accuracy when tested with 384000 testing data.

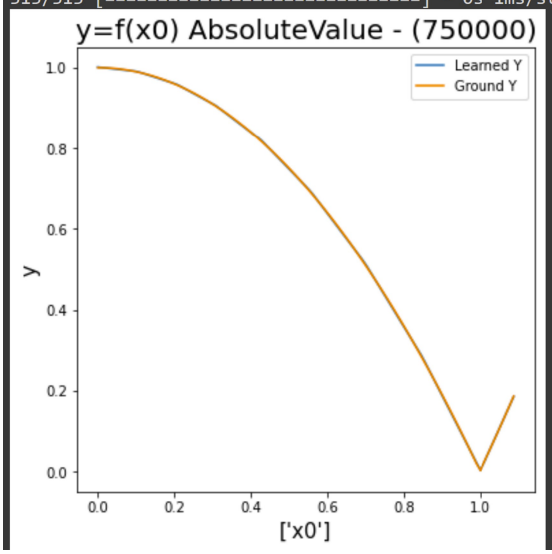
```
sw = StopWatch('Plotting AP vs loss')
plotAP(True,"nItemsTrain", [384000, 750000, 1000000], [['nHidden', 10], ]) # testing the accuracy for different training datas
sw.end()
```

```
Plotting AP vs loss stop watch start time: 2021-12-02 21:40:20.925501
Build for nItemsTrain = 384000
Epoch 1/5
3000/3000 [==============================] - 4s 1ms/step - loss: 0.0390 - mae: 0.0829
Epoch 2/5
3000/3000 [==============================] - 4s 1ms/step - loss: 0.0038 - mae: 0.0312
Epoch 3/5
3000/3000 [==============================] - 4s 1ms/step - loss: 3.8500e-04 - mae: 0.0138
Epoch 4/5
3000/3000 [==============================] - 4s 1ms/step - loss: 1.3658e-04 - mae: 0.0091
Epoch 5/5
3000/3000 [==============================] - 4s 1ms/step - loss: 1.1098e-04 - mae: 0.0083
313/313 [==============================] - 0s 1ms/step - loss: 9.8464e-05 - mae: 0.0084
```



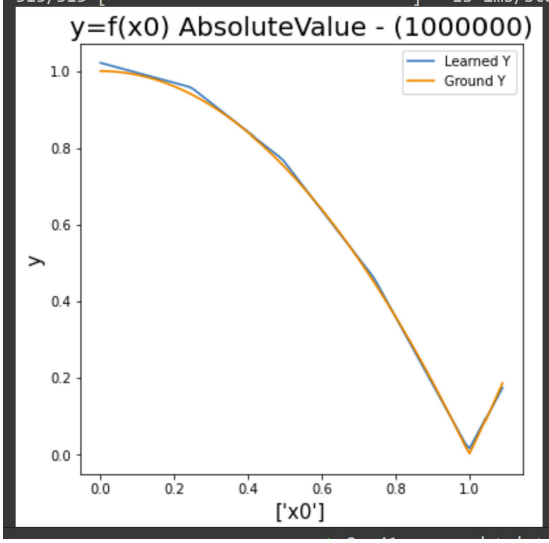
The performance is significantly increased when the training data is double to 750000. In terms of the retail store, this means that the accuracy of predicting the sweatness of strawberries throughout the year is very accurate and precise.

```
Build for nItemsTrain = 750000
Epoch 1/5
5860/5860 [==============================] - 9s 1ms/step - loss: 0.0093 - mae: 0.0418
Epoch 2/5
5860/5860 [==============================] - 8s 1ms/step - loss: 5.2291e-05 - mae: 0.0057
Epoch 3/5
5860/5860 [==============================] - 8s 1ms/step - loss: 4.3962e-05 - mae: 0.0053
Epoch 4/5
5860/5860 [==============================] - 8s 1ms/step - loss: 4.1907e-05 - mae: 0.0051
Epoch 5/5
5860/5860 [==============================] - 8s 1ms/step - loss: 4.0997e-05 - mae: 0.0051
313/313 [==============================] - 0s 1ms/step - loss: 1.7204e-05 - mae: 0.0035
```

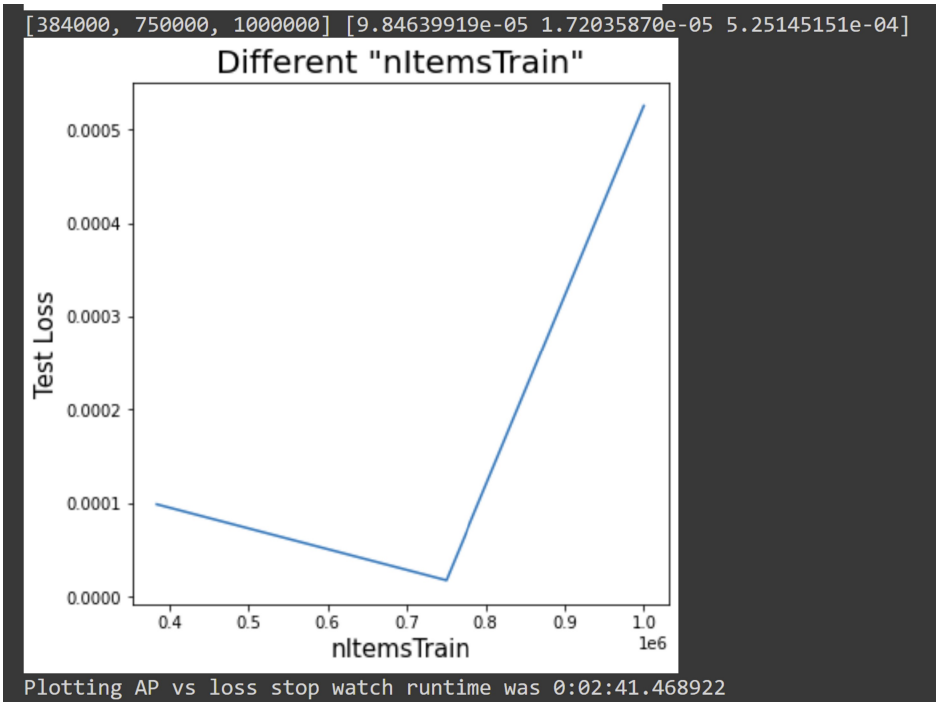


Very surprisingly, the performance drops noticeably when the training data is increased to 1000000 making it less precise specially in the beginning months of the year.

```
Build for nItemsTrain = 1000000
Epoch 1/5
7813/7813 [==============================] - 12s 1ms/step - loss: 0.0195 - mae: 0.0527
Epoch 2/5
7813/7813 [==============================] - 12s 1ms/step - loss: 0.0026 - mae: 0.0260
Epoch 3/5
7813/7813 [==============================] - 11s 1ms/step - loss: 9.2064e-04 - mae: 0.0197
Epoch 4/5
7813/7813 [==============================] - 11s 1ms/step - loss: 5.2882e-04 - mae: 0.0172
Epoch 5/5
7813/7813 [==============================] - 12s 1ms/step - loss: 3.8937e-04 - mae: 0.0158
313/313 [==============================] - 1s 1ms/step - loss: 5.2515e-04 - mae: 0.0184
```



The summary of the precision of the learning graphs with 384000, 750000 and 1000000 is given below consistent to the explanation above.



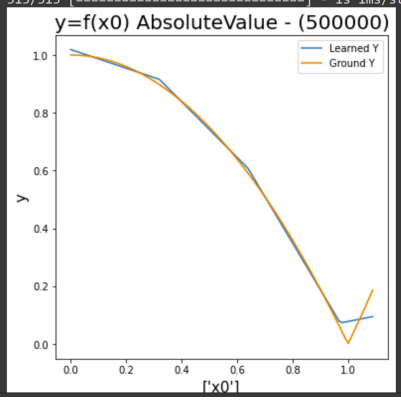
Analysing the running time of the function when the number of <u>Training data</u> is increased:

When the training data is 50000, adding the running time of for each Epoch gives us around 32 seconds.

7s + 6s + 6s + 6s + 6s + 1s = 32s

```
sw = StopWatch('Plotting AP vs loss')
plotAP(True,"nItemsTrain", [500000, 1000000], [['nHidden', 10], ]) # testing the accuracy for different training datas
sw.end()
```

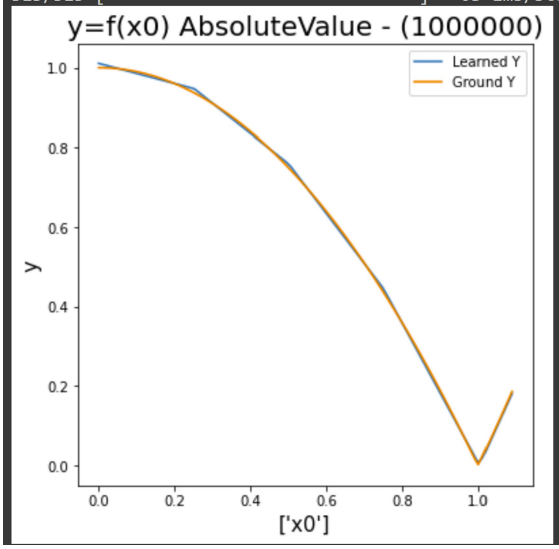
```
Plotting AP vs loss stop watch start time: 2021-12-02 21:47:34.004739
Build for nItemsTrain = 500000
Epoch 1/5
3907/3907 [==============================] - 7s 2ms/step - loss: 0.0590 - mae: 0.0971
Epoch 2/5
3907/3907 [==============================] - 6s 2ms/step - loss: 0.0062 - mae: 0.0383
Epoch 3/5
3907/3907 [==============================] - 6s 2ms/step - loss: 0.0055 - mae: 0.0378
Epoch 4/5
3907/3907 [==============================] - 6s 2ms/step - loss: 0.0051 - mae: 0.0379
Epoch 5/5
3907/3907 [==============================] - 6s 1ms/step - loss: 0.0045 - mae: 0.0374
313/313 [==============================] - 1s 1ms/step - loss: 0.0038 - mae: 0.0348
```



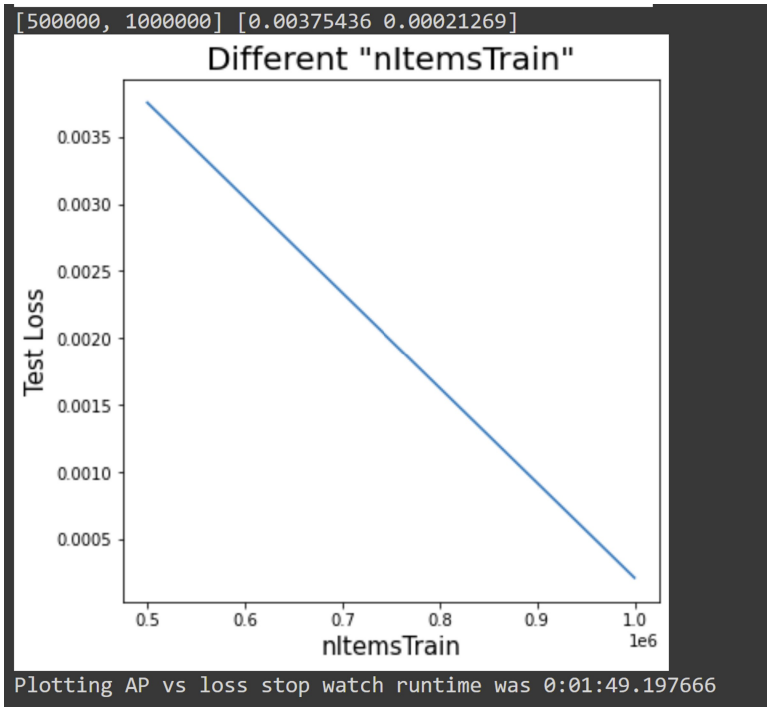
As expected, doubling the size of the training data, doubles the running time. That is to say, it increases the running time of the program to 64 seconds.

12s + 11s + 11s + 11s + 11s = 64s

```
Build for nItemsTrain = 1000000
Epoch 1/5
7813/7813 [==============================] - 12s 1ms/step - loss: 0.0167 - mae: 0.0391
Epoch 2/5
7813/7813 [==============================] - 11s 1ms/step - loss: 2.1823e-04 - mae: 0.0125
Epoch 3/5
7813/7813 [==============================] - 11s 1ms/step - loss: 2.1718e-04 - mae: 0.0125
Epoch 4/5
7813/7813 [==============================] - 11s 1ms/step - loss: 2.1662e-04 - mae: 0.0124
Epoch 5/5
7813/7813 [==============================] - 11s 1ms/step - loss: 2.1634e-04 - mae: 0.0124
313/313 [==============================] - 0s 1ms/step - loss: 2.1269e-04 - mae: 0.0129
```



The graph below illustrates the indirect proportion of the number of testing data with running time to predict the accuracy of the learning function. The total loss of the learning function decreases dramatically when the number of training data is doubled.



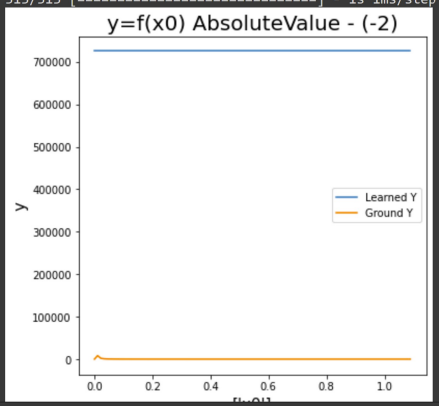
Reporting the accuracy absoluteValue function with negative degrees:

Trying the function $|x^2 - 1|$ with degrees -2, -25 and -50.

If we plot the function $|x^{-2} - 1|$ the outcoming graph does a terrible job in predicting the sweetness of the strawberries coming no where close to the ground function.

```
sw = StopWatch('Plotting AP vs loss')
setAP(["maxX", 1.1])
plotAP(True,"degree",[-2, -25, -50], [['function', "AbsoluteValue"], ]) # Comparing degree -2, -25 and -50
sw.end()
```

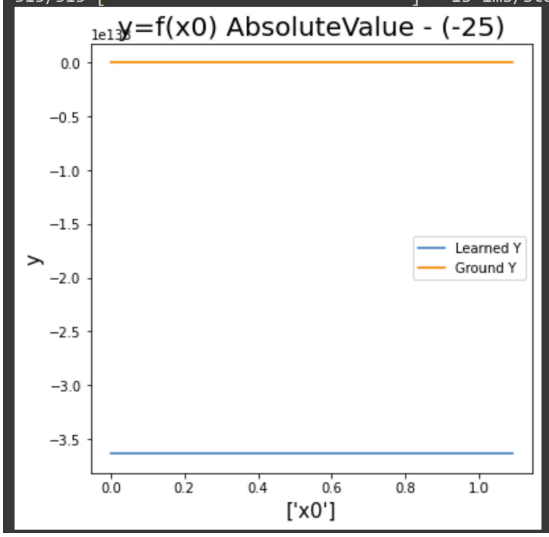
```
Plotting AP vs loss stop watch start time: 2021-12-02 21:56:12.139528
Build for degree = -2
Epoch 1/5
7813/7813 [==============================] - 15s 2ms/step - loss: 1.0001 - mae: 0.0054
Epoch 2/5
7813/7813 [==============================] - 14s 2ms/step - loss: 1.0000 - mae: 0.0021
Epoch 3/5
7813/7813 [==============================] - 14s 2ms/step - loss: 1.0000 - mae: 0.0020
Epoch 4/5
7813/7813 [==============================] - 14s 2ms/step - loss: 1.0000 - mae: 0.0021
Epoch 5/5
7813/7813 [==============================] - 14s 2ms/step - loss: 1.0000 - mae: 0.0020
313/313 [==============================] - 1s 1ms/step - loss: 1.1937e-06 - mae: 8.7688e-04
```



Similar performances can be seen with degree -25 and a better one when degree is -50.

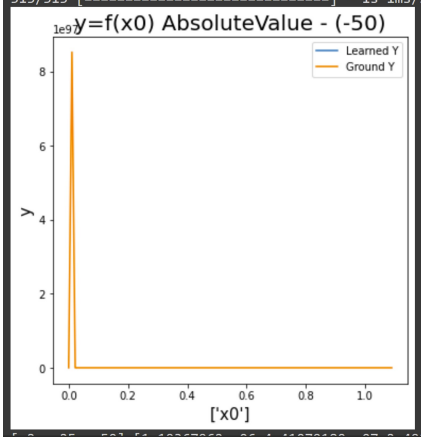
$$\left|x^{-25} - 1\right|$$

```
Build for degree = -25
Epoch 1/5
7813/7813 [==============================] - 16s 2ms/step - loss: 1.0001 - mae: 0.0045
Epoch 2/5
7813/7813 [==============================] - 16s 2ms/step - loss: 1.0000 - mae: 0.0015
Epoch 3/5
7813/7813 [==============================] - 16s 2ms/step - loss: 1.0000 - mae: 0.0015
Epoch 4/5
7813/7813 [==============================] - 17s 2ms/step - loss: 1.0000 - mae: 0.0014
Epoch 5/5
7813/7813 [==============================] - 17s 2ms/step - loss: 1.0000 - mae: 0.0015
313/313 [==============================] - 1s 1ms/step - loss: 4.4108e-07 - mae: 5.0427e-04
```



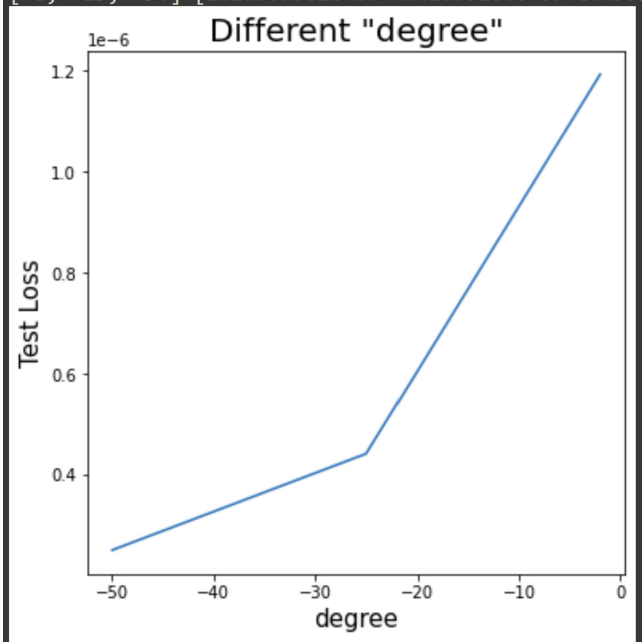
$|x^{-50} - 1|$

```
Build for degree = -50
/usr/local/lib/python3.7/dist-packages/numpy/core/_methods.py:205: RuntimeWarning: overflow encountered in multiply
  x = um.multiply(x, x, out=x)
Epoch 1/5
7813/7813 [==============================] - 18s 2ms/step - loss: 4.0240e-05 - mae: 0.0034
Epoch 2/5
7813/7813 [==============================] - 16s 2ms/step - loss: 2.4994e-07 - mae: 4.9996e-04
Epoch 3/5
7813/7813 [==============================] - 16s 2ms/step - loss: 2.4994e-07 - mae: 4.9996e-04
Epoch 4/5
7813/7813 [==============================] - 15s 2ms/step - loss: 2.4994e-07 - mae: 4.9996e-04
Epoch 5/5
7813/7813 [==============================] - 15s 2ms/step - loss: 2.4994e-07 - mae: 4.9996e-04
313/313 [==============================] - 1s 1ms/step - loss: 2.4995e-07 - mae: 4.9995e-04
```



Here's the a direct comparison of their total loss in one picture:

```
[-2, -25, -50] [1.19367962e-06 4.41079180e-07 2.49951029e-07]
```




```
Plotting AP vs loss stop watch runtime was 0:04:25.445913
```

What if a second dimension is added to add more meaning and how the performance of the learning

graph changes accordingly:

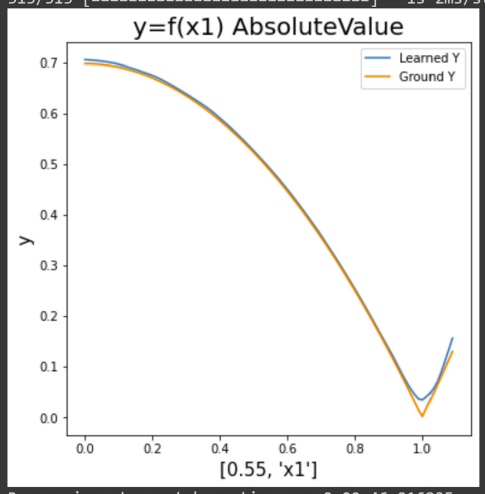
As reported before, the first dimension represents the sweetness of the strawberry depending on the quantity of supplies and the time of the year, while, the second dimension represents the colour of the strawberry depending on the quantity of supplies and the time of the year.

We also change the value of the parameter "graph_r" from 0 to 1 so the graph only represents the data about the colour of the strawberry.

Even though the number of dimensions was increased, the outcoming graph did a great job in predicting the colour of the strawberry when the degree of the absoluteValue function was 2.

```
sw = StopWatch('Regression')
# Graph 1 # Report on running time
all(["nDimensions", 2], ["graph_r", 1], ["function","AbsoluteValue"], ["maxX", 1.1], )
# Introducing 2 dimensions x = [x0, x1] -> x0: sweetness, x1: Colour
sw.end()
```

```
Regression stop watch start time: 2021-12-02 22:17:04.186885
Epoch 1/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0085 - mae: 0.0491
Epoch 2/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0017 - mae: 0.0274
Epoch 3/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0015 - mae: 0.0253
Epoch 4/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0012 - mae: 0.0228
Epoch 5/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0011 - mae: 0.0213
313/313 [==============================] - 1s 2ms/step - loss: 0.0011 - mae: 0.0240
```




```
Regression stop watch runtime was 0:00:46.216935
```

If we categorize our x1 input values into 4 categories of [0.0 (Jan)- 0.3 (April)], [0.3 (April)- 0.6 (Jul)], [0.6 (Jul)- 0.9 (Oct)], [0.9 (Oct) - 1.1 (Dec)], we get the below categorization.

- In the graph below, note that:
  - Y = -0.04 represents strawberries with brown caps, that are fully raw.
  - Y = 0.00 represents strawberries with green caps, that are partially ripe.
  - Y = +0.04 represents strawberries with red caps, that are fully ripe.

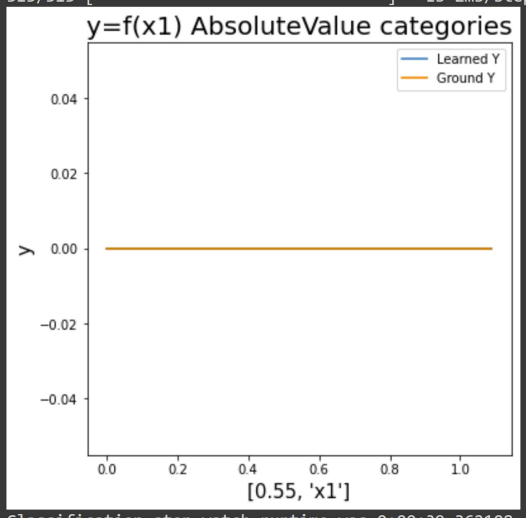
Strangely, despite different categorization, the learning graph predicts that throughout the year

the colour of the strawberries are consistently green (Y = 0.00).

The categorization suggests that in order to fulfill the demand of strawberries in our retail store, we must buy them in small batches every month instead of buying them in bulk. This would ensure that the partially ripe green strawberries have enough time to ripen into vibrant red colours so that they are ready to be sold to the final consumer.

```
sw = StopWatch('Classification')
setAP(["nDimensions", 2], ["graph_r", 1], ["maxX", 1.1])
all(["categorize",[0.91, 0.64, 0.19, 0.21]])
sw.end()
```

```
Classification stop watch start time: 2021-12-02 22:21:26.470983
Epoch 1/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0234 - accuracy: 0.9951
Epoch 2/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0058 - accuracy: 0.9980
Epoch 3/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0048 - accuracy: 0.9981
Epoch 4/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0042 - accuracy: 0.9983
Epoch 5/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0039 - accuracy: 0.9983
313/313 [==============================] - 1s 2ms/step - loss: 0.0085 - accuracy: 0.9957
```



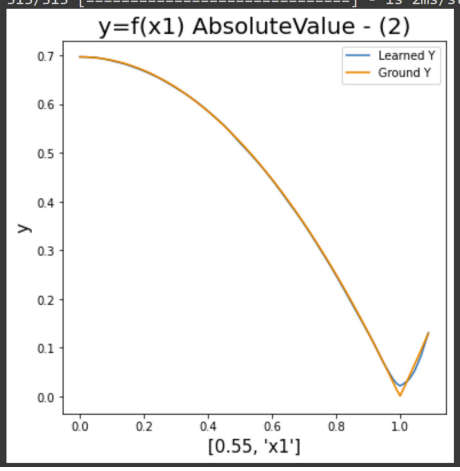

```
Classification stop watch runtime was 0:00:39.362108
```

Here is the direct comparison of the performance of learning graph in predicting the colour of the strawberries when degrees 2, 25 and 50 are used in absoluteValue function graph.

- When the degree is 2, note that the graph learned Y (the predicted colour of the strawberries) is the closest to the graph ground Y (the actual colour of the strawberries).

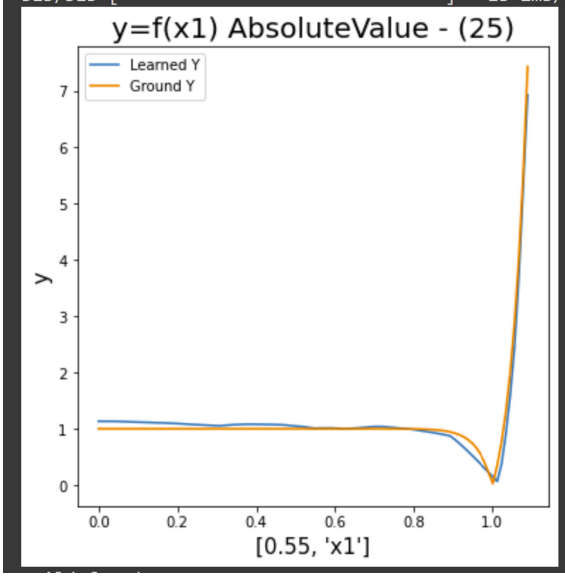
```
sw = StopWatch('Plotting AP vs loss')
setAP(["nDimensions", 2], ["graph_r", 1], ["maxX", 1.1])
plotAP(True,"degree",[2, 25 , 50], [['function', "AbsoluteValue"], ]) # Comparing degree 2, 25 and 50
sw.end()
```

```
Plotting AP vs loss stop watch start time: 2021-12-02 22:22:06.606003
Build for degree = 2
Epoch 1/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0076 - mae: 0.0495
Epoch 2/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0017 - mae: 0.0279
Epoch 3/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0015 - mae: 0.0255
Epoch 4/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0012 - mae: 0.0235
Epoch 5/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0011 - mae: 0.0217
313/313 [==============================] - 1s 2ms/step - loss: 2.4383e-04 - mae: 0.0096
```



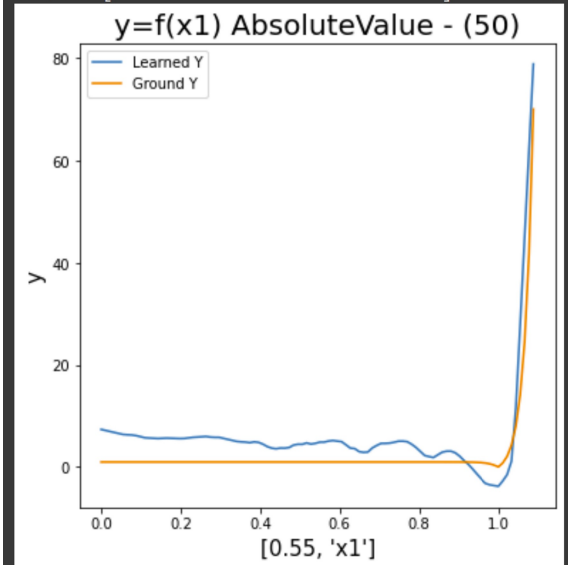
- When the degree is 25, note that the graph learned Y (the predicted colour of the strawberries) is somewhat close to the graph ground Y (the actual colour of the strawberries).

```
Build for degree = 25
Epoch 1/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.5157 - mae: 0.2164
Epoch 2/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.2916 - mae: 0.1243
Epoch 3/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.2528 - mae: 0.1165
Epoch 4/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.2220 - mae: 0.1085
Epoch 5/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.1929 - mae: 0.1029
313/313 [==============================] - 1s 2ms/step - loss: 0.1742 - mae: 0.0878
```

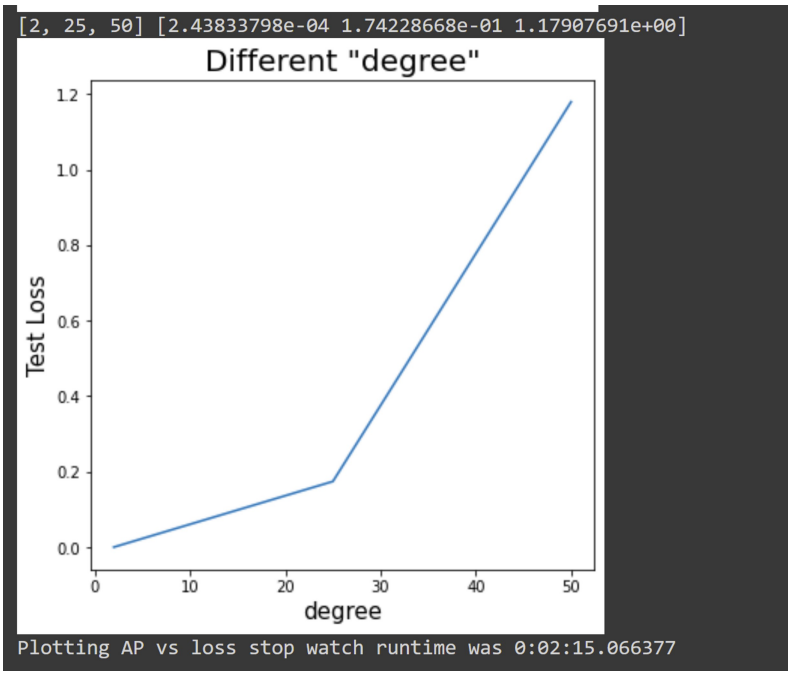


- When the degree is 50, note that the graph learned Y (the predicted colour of the strawberries) is the least closest to the graph ground Y (the actual colour of the strawberries).

```
Build for degree = 50
Epoch 1/5
3000/3000 [==============================] - 8s 2ms/step - loss: 0.9146 - mae: 0.1066
Epoch 2/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.8332 - mae: 0.1037
Epoch 3/5
3000/3000 [==============================] - 8s 3ms/step - loss: 0.7626 - mae: 0.1019
Epoch 4/5
3000/3000 [==============================] - 7s 2ms/step - loss: 0.7012 - mae: 0.1004
Epoch 5/5
3000/3000 [==============================] - 8s 3ms/step - loss: 0.6461 - mae: 0.0976
313/313 [==============================] - 1s 2ms/step - loss: 1.1791 - mae: 0.0878
```



The graph below confirms the mentioned points about the test loss of the learning graph (the predicted colour of the strawberries) compared to the actual graph (the actual colour of the strawberries).

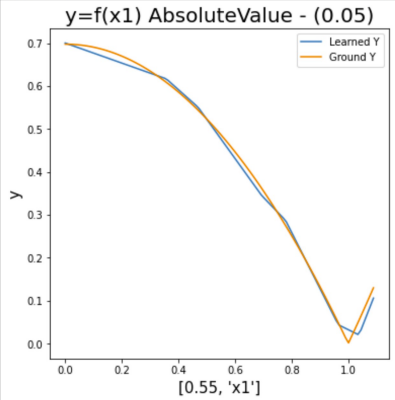


Testing the performance of the learner graph when noise is added to the training data

- If we add a noise of 0.05 to the learner graph, it still gives us the data that is close to the real data. That is to say, the predicted colour of strawberries compared to how they really are in different seasons.

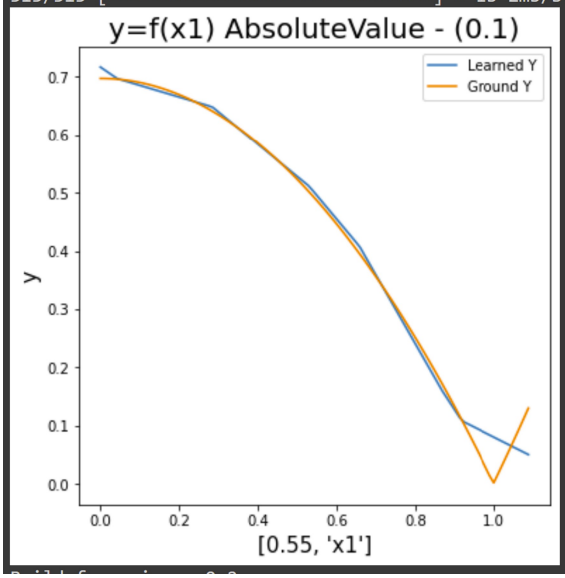
```
sw = StopWatch('Plotting AP vs loss')
plotAP(True,"noise", [0.05, 0.1, 0.2], [['nHidden', 10], ]) # testing the accuracy for different training datas
sw.end()
```

```
Plotting AP vs loss stop watch start time: 2021-12-02 22:55:01.959787
Build for noise = 0.05
Epoch 1/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0672 - mae: 0.1473
Epoch 2/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0103 - mae: 0.0801
Epoch 3/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0089 - mae: 0.0753
Epoch 4/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0085 - mae: 0.0740
Epoch 5/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0079 - mae: 0.0710
313/313 [==============================] - 1s 2ms/step - loss: 0.0025 - mae: 0.0388
```



- Increasing the noise to 0.1 effects the accuracy of the prediction marginally until the month of October (0,9) before being completely off in the last two months.

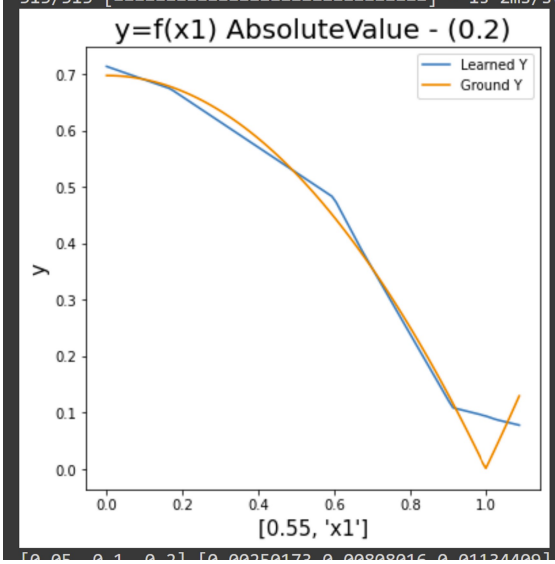
```
Build for noise = 0.1
Epoch 1/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0739 - mae: 0.1849
Epoch 2/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0302 - mae: 0.1378
Epoch 3/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0285 - mae: 0.1345
Epoch 4/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0274 - mae: 0.1321
Epoch 5/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0266 - mae: 0.1303
313/313 [==============================] - 1s 2ms/step - loss: 0.0081 - mae: 0.0556
```



- Doubling the amount of noise from 0.1 to 0.2 only gives accurate information about the color of

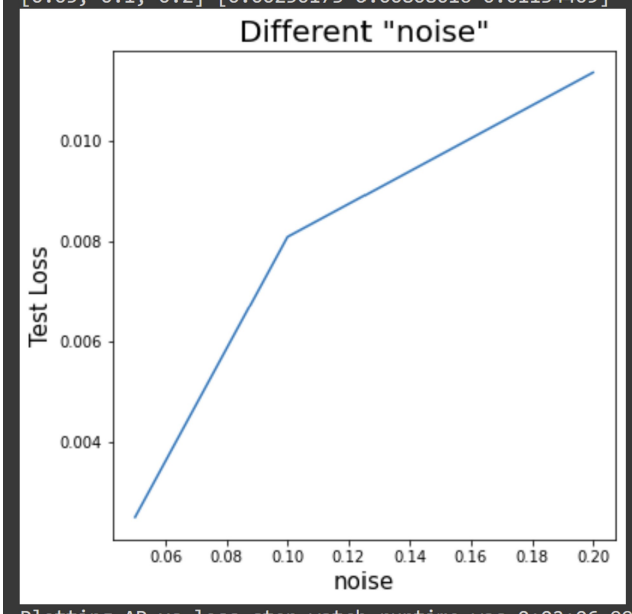
the strawberries in August (0.7), September (0.8) and October (0.9) and giving considerably inaccurate data in the rest of the months.

```
Build for noise = 0.2
Epoch 1/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.1181 - mae: 0.2626
Epoch 2/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0804 - mae: 0.2296
Epoch 3/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0794 - mae: 0.2282
Epoch 4/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0789 - mae: 0.2276
Epoch 5/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.0787 - mae: 0.2274
313/313 [==============================] - 1s 2ms/step - loss: 0.0113 - mae: 0.0778
```



Below is the direct comparision of Test Loss with respect to the level of noise:

```
[0.05, 0.1, 0.2] [0.00250173 0.00808016 0.01134409]
```



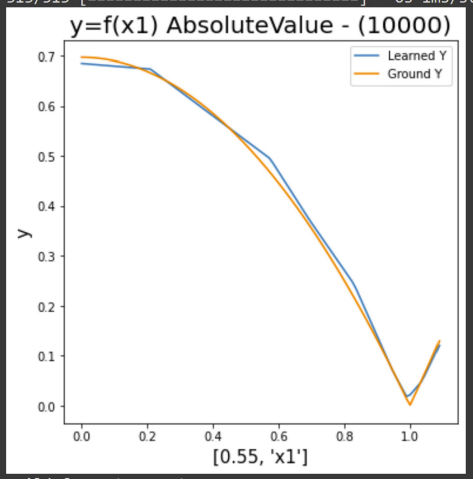

```
Plotting AP vs loss stop watch runtime was 0:02:06.997712
```

Will changing the testing data change anything?

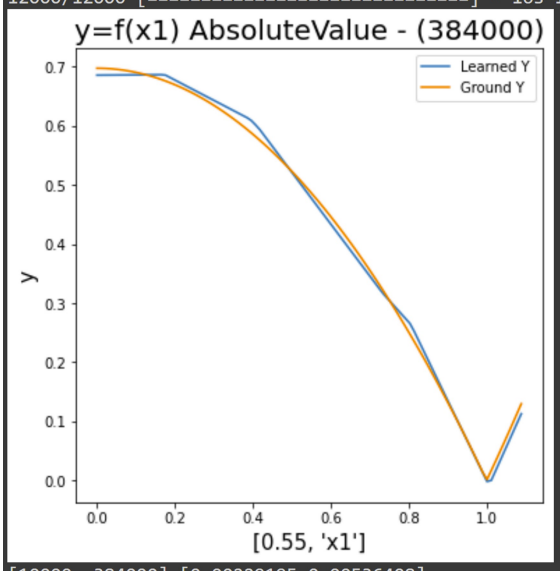
Having the default parameters and only changing the testing data does not affect the performance of the learned graph any better or worse. Below are the graphs drawn with 10000 and 384000 testing data respectively .

```
sw = StopWatch('Plotting AP vs loss')
plotAP(True,"nItemsTest", [10000, 384000], [['nHidden', 10], ]) # Comparing different testing datas
sw.end()
```

```
Plotting AP vs loss stop watch start time: 2021-12-02 23:05:41.277408
Build for nItemsTest = 10000
Epoch 1/5
3000/3000 [==============================] - 5s 1ms/step - loss: 0.0444 - mae: 0.1197
Epoch 2/5
3000/3000 [==============================] - 4s 1ms/step - loss: 0.0101 - mae: 0.0695
Epoch 3/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0064 - mae: 0.0539
Epoch 4/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0046 - mae: 0.0484
Epoch 5/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.0032 - mae: 0.0415
313/313 [==============================] - 0s 1ms/step - loss: 0.0023 - mae: 0.0359
```



```
Build for nItemsTest = 384000
Epoch 1/5
3000/3000 [==============================] - 5s 1ms/step - loss: 0.0447 - mae: 0.1178
Epoch 2/5
3000/3000 [==============================] - 4s 1ms/step - loss: 0.0083 - mae: 0.0607
Epoch 3/5
3000/3000 [==============================] - 4s 1ms/step - loss: 0.0058 - mae: 0.0493
Epoch 4/5
3000/3000 [==============================] - 4s 1ms/step - loss: 0.0055 - mae: 0.0492
Epoch 5/5
3000/3000 [==============================] - 4s 1ms/step - loss: 0.0055 - mae: 0.0495
12000/12000 [==============================] - 16s 1ms/step - loss: 0.0054 - mae: 0.0493
```



Function 2:

Let the Covid-19 function be $y = \frac{(x-1)^2}{(x^2-1)} * C$ where C represents a constant arbitary number between a range depending on the value of the parameter "maxX".

Let parameter "maxX" represent the number of days since the start of Covid-19 pandemic in Ontario.
Let y represent the total number of cases reported on each day.

Our function Covid-19 has been calculated in such way that for different ranges (days) a random number of cases is being generated based on the input "maxX"= The total number of days in the report.

```
# Function Covid-19
import random

if AP["function"]=="Covid-19":
  y = 1
  for i in range(AP["nDimensions"]):
    for m in range(0, int((AP["maxX"] - AP["maxX"]/2.5)), 5):
      if (int(AP["maxX"] - AP["maxX"]/5)) <= m <= int((AP["maxX"] - AP["maxX"]/2.5)):
        if (((x[i] - 1) ** 2)/(x**2 - 1)) * random.randint(0, int((AP["maxX"] - AP["maxX"]/5))) < 0:
          pass
        else:
          y *= (((x[i] - 1) ** 2)/(x**2 - 1)) * random.randint(0, int((AP["maxX"] - AP["maxX"]/5)))

      else:
        if (((x[i] - 1) ** 2)/(x**2 - 1)) * random.randint(0, int((AP["maxX"] - AP["maxX"]/5))) < 0:
          pass
        else:
          y *= (((x[i] - 1) ** 2)/(x**2 - 1)) * random.randint(0, int((AP["maxX"] - AP["maxX"]/2)))

    for m in range(int((AP["maxX"] - AP["maxX"]/2.5)), AP["maxX"], 5):
      if (((x[i] - 1) ** 2)/(x**2 - 1)) * random.randint(0, int((AP["maxX"] - AP["maxX"]/5))) < 0:
        pass
      else:
        y *= (((x[i] - 1) ** 2)/(x**2 - 1)) * random.randint(0, int((AP["maxX"] - AP["maxX"]/10)))

  return y
```

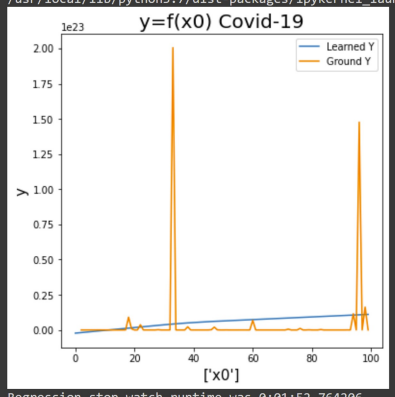
Lets analyse the real data that has been reported and how well the leaning graph has done in predicting the number of cases each day.

The actual data for 100 days illustrates that for the first 35 days the number Covid cases are consistently low before suddenly rising to an unprecedented peak (2000 positive cases) for two consecutive days. Thereafter, Ontario went under lockdown for couple of months resulting in Covid cases to fall significantly. Once the restrictions were loosened, the second wave brought about a rise in Covid cases to about 1500 positive cases.

The learning graph on the other side, is predicting the number of positive cases closely except during the two major waves.

```
sw = StopWatch('Regression')
all(["function","Covid-19"], ["maxX", 100], ) # maxX = 100 representing the number of days
sw.end()
```

```
Regression stop watch start time: 2021-12-04 03:42:32.624856
Epoch 1/5
3000/3000 [==============================] - 6s 2ms/step - loss: 1.0002 - mae: 0.0686
Epoch 2/5
3000/3000 [==============================] - 6s 2ms/step - loss: 1.0000 - mae: 0.0667
Epoch 3/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.9998 - mae: 0.0665
Epoch 4/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.9998 - mae: 0.0667
Epoch 5/5
3000/3000 [==============================] - 6s 2ms/step - loss: 0.9998 - mae: 0.0665
313/313 [==============================] - 1s 1ms/step - loss: 0.2226 - mae: 0.0691
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:42: RuntimeWarning: invalid value encountered in true_divide
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:40: RuntimeWarning: invalid value encountered in true_divide
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:45: RuntimeWarning: invalid value encountered in true_divide
```



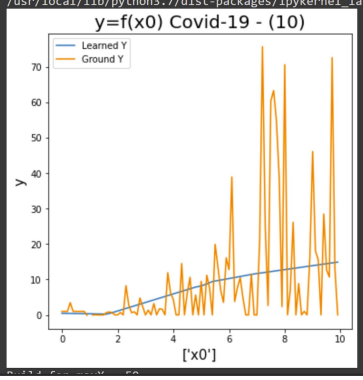

```
Regression stop watch runtime was 0:01:52.764206
```

Lets review the data closely for the first 10 days, 50 days and 100 days respectively.

For the first 10 days, the positive cases were rising rapidly. Whereas the learning graph predicates the total cases to increase gradually giving us less accurate information.

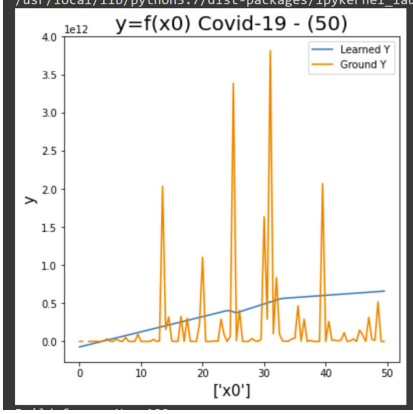
```
sw = StopWatch('Plotting AP vs loss')

plotAP(True,"maxX", [10, 50 ,100], [['nHidden', 10], ]) # Comparing the first 10, 50 and 70 days of the number of positive cases
sw.end()
```

```
Plotting AP vs loss stop watch start time: 2021-12-04 04:23:28.620743
Build for maxX = 10
Epoch 1/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.8727 - mae: 0.5770
Epoch 2/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.8643 - mae: 0.5698
Epoch 3/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.8643 - mae: 0.5690
Epoch 4/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.8643 - mae: 0.5693
Epoch 5/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.8643 - mae: 0.5691
313/313 [==============================] - 1s 1ms/step - loss: 0.8514 - mae: 0.5600
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:46: RuntimeWarning: invalid value encountered in true_divide
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:49: RuntimeWarning: invalid value encountered in true_divide
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:52: RuntimeWarning: invalid value encountered in true_divide
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:55: RuntimeWarning: invalid value encountered in true_divide
```



For the first 50 days, the positive cases were oscillating in an inconsistent manner. Similar, like the first 10 days, the learning graph predicates the total cases to increase gradually giving us less accurate information.

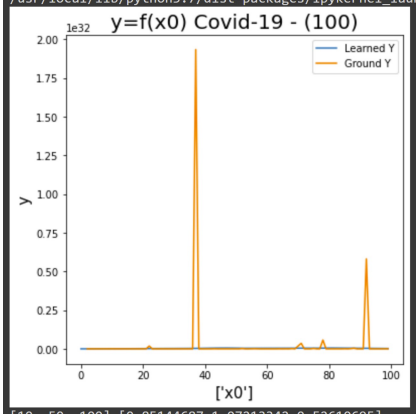
```
Build for maxX = 50
Epoch 1/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.9904 - mae: 0.2900
Epoch 2/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.9881 - mae: 0.2874
Epoch 3/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.9881 - mae: 0.2872
Epoch 4/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.9881 - mae: 0.2874
Epoch 5/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.9881 - mae: 0.2878
313/313 [==============================] - 1s 1ms/step - loss: 1.0721 - mae: 0.2935
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:46: RuntimeWarning: invalid value encountered in true_divide
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:49: RuntimeWarning: invalid value encountered in true_divide
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:52: RuntimeWarning: invalid value encountered in true_divide
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:55: RuntimeWarning: invalid value encountered in true_divide
```



For the first 100 days, two major waves occurred. The highest was between days 35 - 40 recording almost 2000 positive cases, while the second highest wave was between days 87 - 92 recording about 500 positive cases.

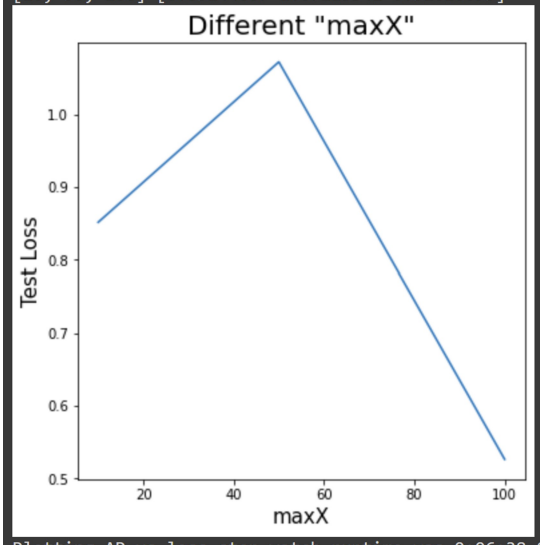
The learning graph on the other side, is predicting the number of positive cases closely except during the two major waves.

```
Build for maxX = 100
Epoch 1/5
3000/3000 [==============================] - 5s 2ms/step - loss: 1.0017 - mae: 0.0693
Epoch 2/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.9997 - mae: 0.0647
Epoch 3/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.9997 - mae: 0.0645
Epoch 4/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.9997 - mae: 0.0647
Epoch 5/5
3000/3000 [==============================] - 5s 2ms/step - loss: 0.9998 - mae: 0.0643
313/313 [==============================] - 1s 1ms/step - loss: 0.5261 - mae: 0.0606
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:46: RuntimeWarning: invalid value encountered in true_divide
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:49: RuntimeWarning: invalid value encountered in true_divide
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:52: RuntimeWarning: invalid value encountered in true_divide
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:55: RuntimeWarning: invalid value encountered in true_divide
```



We can note that the total test loss in the first 10 days was reported around 0.85 which is moderately accurate compared to the other two; while the total test loss in the first 50 days was reported around 1.07 making it the least accurate and reliable. Finally, the total test loss in the first 100 days was reported around 0.52 making it the most accurate and reliable out of the 3 Covid-19 graphs.

```
[10, 50, 100] [0.85144687 1.07213342 0.52610695]
```




```
Plotting AP vs loss stop watch runtime was 0:06:38.072641
```