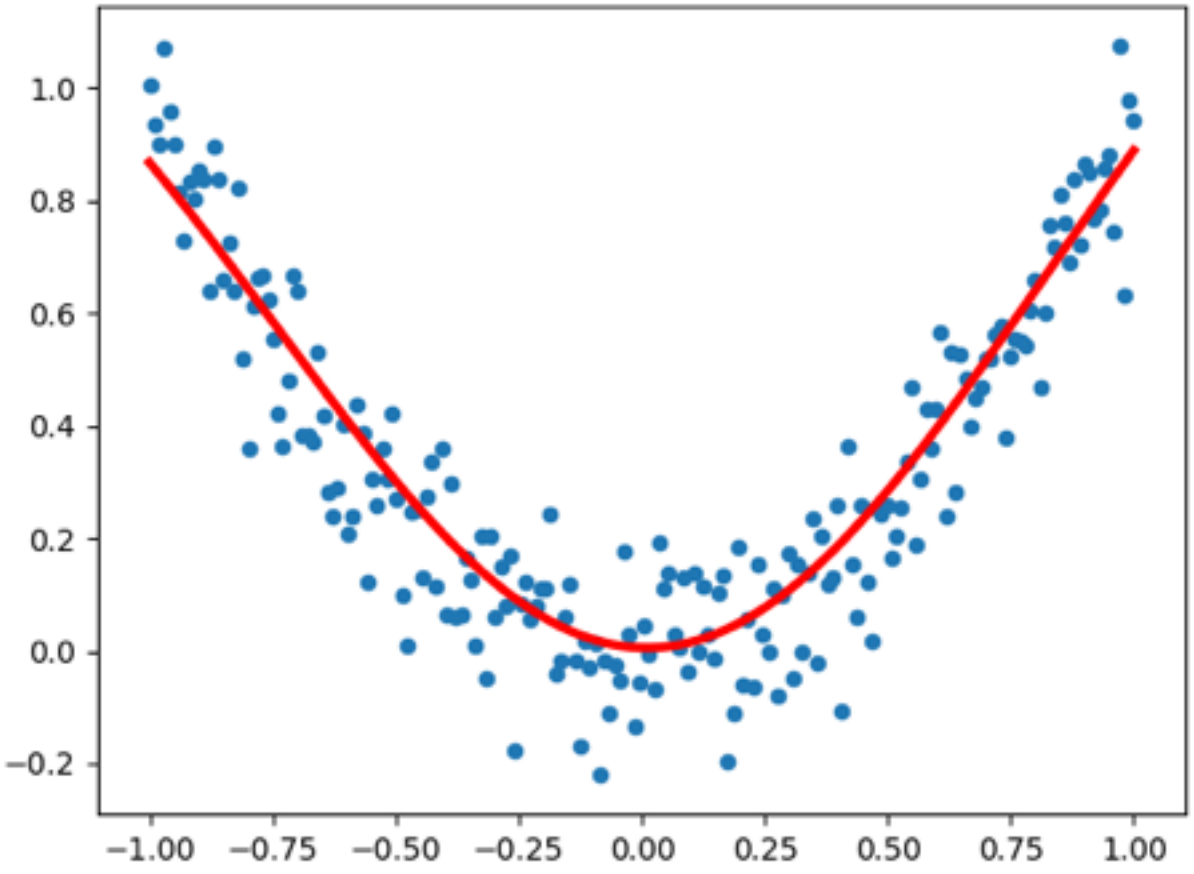
Data_Function



Prime_net_weights

```
xinyuem@asb9700u-c02:~/sfuhome/Assignment4$ python prime_classifier.py
[Epoch 0]:      validation loss: 0.20127979,     validation accuracy: 69.43%
[Epoch 1]:      validation loss: 0.11581653,     validation accuracy: 85.94%
[Epoch 2]:      validation loss: 0.09948711,     validation accuracy: 87.44%
[Epoch 3]:      validation loss: 0.09273907,     validation accuracy: 88.30%
[Epoch 4]:      validation loss: 0.08897993,     validation accuracy: 88.72%
[Epoch 5]:      validation loss: 0.08652826,     validation accuracy: 89.06%
[Epoch 6]:      validation loss: 0.08472717,     validation accuracy: 89.32%
[Epoch 7]:      validation loss: 0.08315523,     validation accuracy: 89.39%
[Epoch 8]:      validation loss: 0.08117772,     validation accuracy: 89.68%
[Epoch 9]:      validation loss: 0.07827912,     validation accuracy: 90.09%
[Epoch 10]:     validation loss: 0.07484797,     validation accuracy: 90.62%
[Epoch 11]:     validation loss: 0.07118534,     validation accuracy: 91.22%
[Epoch 12]:     validation loss: 0.06739957,     validation accuracy: 91.82%
[Epoch 13]:     validation loss: 0.06351675,     validation accuracy: 92.39%
[Epoch 14]:     validation loss: 0.05953572,     validation accuracy: 92.90%
[Epoch 15]:     validation loss: 0.05560661,     validation accuracy: 93.60%
[Epoch 16]:     validation loss: 0.05196045,     validation accuracy: 94.12%
[Epoch 17]:     validation loss: 0.04873159,     validation accuracy: 94.46%
[Epoch 18]:     validation loss: 0.04594967,     validation accuracy: 94.72%
[Epoch 19]:     validation loss: 0.04357872,     validation accuracy: 94.92%
[Epoch 20]:     validation loss: 0.04155667,     validation accuracy: 95.21%
[Epoch 21]:     validation loss: 0.03982112,     validation accuracy: 95.45%
[Epoch 22]:     validation loss: 0.03831892,     validation accuracy: 95.65%
[Epoch 23]:     validation loss: 0.03700730,     validation accuracy: 95.79%
[Epoch 24]:     validation loss: 0.03585224,     validation accuracy: 95.96%
[Epoch 25]:     validation loss: 0.03482659,     validation accuracy: 96.07%
[Epoch 26]:     validation loss: 0.03390858,     validation accuracy: 96.12%
[Epoch 27]:     validation loss: 0.03308090,     validation accuracy: 96.21%
[Epoch 28]:     validation loss: 0.03232983,     validation accuracy: 96.32%
[Epoch 29]:     validation loss: 0.03164452,     validation accuracy: 96.39%
[Epoch 30]:     validation loss: 0.03101625,     validation accuracy: 96.47%
[Epoch 31]:     validation loss: 0.03043786,     validation accuracy: 96.51%
[Epoch 32]:     validation loss: 0.02990342,     validation accuracy: 96.58%
[Epoch 33]:     validation loss: 0.02940791,     validation accuracy: 96.62%
[Epoch 34]:     validation loss: 0.02894711,     validation accuracy: 96.68%
[Epoch 35]:     validation loss: 0.02851736,     validation accuracy: 96.73%
[Epoch 36]:     validation loss: 0.02811557,     validation accuracy: 96.75%
[Epoch 37]:     validation loss: 0.02773900,     validation accuracy: 96.78%
[Epoch 38]:     validation loss: 0.02738531,     validation accuracy: 96.83%
[Epoch 39]:     validation loss: 0.02705240,     validation accuracy: 96.88%
[Epoch 40]:     validation loss: 0.02673845,     validation accuracy: 96.93%
[Epoch 41]:     validation loss: 0.02644183,     validation accuracy: 96.95%
[Epoch 42]:     validation loss: 0.02616109,     validation accuracy: 96.98%
[Epoch 43]:     validation loss: 0.02589496,     validation accuracy: 96.99%
[Epoch 44]:     validation loss: 0.02564228,     validation accuracy: 97.04%
[Epoch 45]:     validation loss: 0.02540203,     validation accuracy: 97.04%
[Epoch 46]:     validation loss: 0.02517329,     validation accuracy: 97.03%
[Epoch 47]:     validation loss: 0.02495521,     validation accuracy: 97.04%
[Epoch 48]:     validation loss: 0.02474705,     validation accuracy: 97.06%
[Epoch 49]:     validation loss: 0.02454812,     validation accuracy: 97.08%
```

Simple_net_weight

```
xinyuem@asb9700u-c02:~/sfuhome/Assignment4$ python toy_example_regressor.py
[Epoch 0]: loss: 11.849840680207045
[Epoch 0]: loss: 8.497981890189003
[Epoch 0]: loss: 5.328049060438643
[Epoch 50]: loss: 0.09140990585249589
[Epoch 50]: loss: 0.06283183778140382
[Epoch 50]: loss: 0.10397231051365105
[Epoch 100]: loss: 0.08351043968593128
[Epoch 100]: loss: 0.05309422256241874
[Epoch 100]: loss: 0.08520996548332978
[Epoch 150]: loss: 0.07758266991455717
[Epoch 150]: loss: 0.048325053942393295
[Epoch 150]: loss: 0.07726032493362645
[Epoch 200]: loss: 0.07034483556238813
[Epoch 200]: loss: 0.04322272883085659
[Epoch 200]: loss: 0.07005388245330851
[Epoch 250]: loss: 0.06260327416419362
[Epoch 250]: loss: 0.037941043352188786
[Epoch 250]: loss: 0.06273526894497457
[Epoch 300]: loss: 0.0548465230927541
[Epoch 300]: loss: 0.03276873195956387
[Epoch 300]: loss: 0.05546474746756004
[Epoch 350]: loss: 0.047452478331411245
[Epoch 350]: loss: 0.02797068036362642
[Epoch 350]: loss: 0.04854943284639317
[Epoch 400]: loss: 0.04071799052760981
[Epoch 400]: loss: 0.02375269364294281
[Epoch 400]: loss: 0.042262877855952924
[Epoch 450]: loss: 0.03483529869541965
[Epoch 450]: loss: 0.02023478814721664
[Epoch 450]: loss: 0.03678604400341678
[Epoch 500]: loss: 0.029884499581138767
[Epoch 500]: loss: 0.017446934787788054
[Epoch 500]: loss: 0.032193808988814354
[Epoch 550]: loss: 0.025849453195467244
[Epoch 550]: loss: 0.015345025376272577
[Epoch 550]: loss: 0.02846897519896875
[Epoch 600]: loss: 0.022647128783939315
[Epoch 600]: loss: 0.01383710428702269
[Epoch 600]: loss: 0.025530250540547802
[Epoch 650]: loss: 0.020158802776304784
[Epoch 650]: loss: 0.012809927301005022
[Epoch 650]: loss: 0.023262353775251314
[Epoch 700]: loss: 0.018255553827406564
[Epoch 700]: loss: 0.012149685985266659
[Epoch 700]: loss: 0.021540699864438993
[Epoch 750]: loss: 0.016815272052400403
[Epoch 750]: loss: 0.011754912282025994
[Epoch 750]: loss: 0.020247847382716858
[Epoch 800]: loss: 0.015731720100385635
[Epoch 800]: loss: 0.011542400524165861
[Epoch 800]: loss: 0.019282190516149773
[Epoch 850]: loss: 0.014917690496398166
[Epoch 850]: loss: 0.011448183215989859
[Epoch 850]: loss: 0.018560889250202176
[Epoch 900]: loss: 0.014304508796368638
[Epoch 900]: loss: 0.011425656220884718
[Epoch 900]: loss: 0.01801924659323223
[Epoch 950]: loss: 0.013839696185046949
[Epoch 950]: loss: 0.011442471572962144
[Epoch 950]: loss: 0.017608315145293724
Validation Loss 0.011963719361285072
```

Report:

First, need to implement DenseLayer. To do this, need to implement three functions:

DenseLayer

1: computer_activation

-To find output value with the function o = x.w + b

- Solution:
  - add this in into compute_activation(self, x): output = np.dot(x, self.w) + self.b

2: comput_gradient

-add these to the comput_gradient function:

```
temp = self._output_error_gradient

self.dw = np.dot(self._input_data.T, temp)

 tempMatrixRowNo = self._output_error_gradient.shape[0]

tempMatrix = np.ones((1, tempMatrixRowNo))

self.db = np.dot(tempMatrix, temp)

self._input_error_gradient = np.dot(temp, self.w.T)
```

- From the lecture notes, and what I learned in the class. we need to use output compute input

3: update_weight:

-add this into update_weight function:

```
self.w = self.w - learning_rate * self.dw
self.b = self.b - learning_rate * self.db
```

-give the weight to input

NeuralNet

1:compute_activations:

-add these into compute_activations function:

```
output = x
#TODO Compute the loss
for i in self._layers:
output = i.compute_activation(output)

loss = self.loss.compute_activation(output, target)
```

-compute each layer and loss

2: compute_gradients:

-add these into compute_gradients function:

```
self.loss.compute_gradient()
output = self.loss.get_input_error_gradient()
for layer in reversed(self._layers):
    layer.set_output_error_gradient(output)
    layer.compute_gradient()
    output = layer.get_input_error_gradient()
```

-the first layer output its the next layer's input

3:update_weights:

-add these into update_weights function:

```
for layer in self._layers:
    layer.update_weights(learning_rate)
```

-to fix the weight and update it

use code "toy_example_regressor.py" and "prime_classifier.py" get simple_net_weight, data_function and prime_net_weight