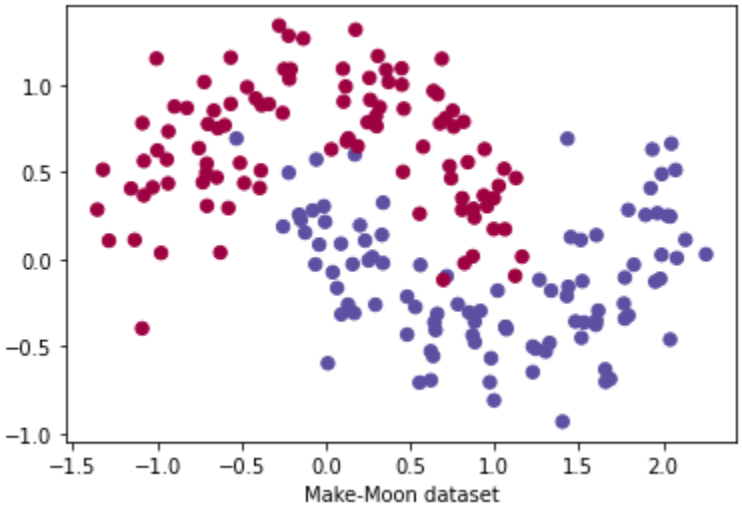
Task1

a) Generate and visualize Make-Moons dataset



b) Derive the activation function

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^{2}(x)$$

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

$$\text{relu}(x) = \max(0, x)$$

$$\frac{d}{dx}\text{relu}(x) = x > 0 \ ? \ 1 : 0$$

c) Build the neural network

d) Backpropagation

$$\delta_t = \frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2}$$

$$\frac{\partial L}{\partial W_2} = a_1^T \delta_t$$

$$\frac{\partial L}{\partial b_2} = \sum \delta_t$$

$$\delta_2 = diff * \delta_t W_2^T$$

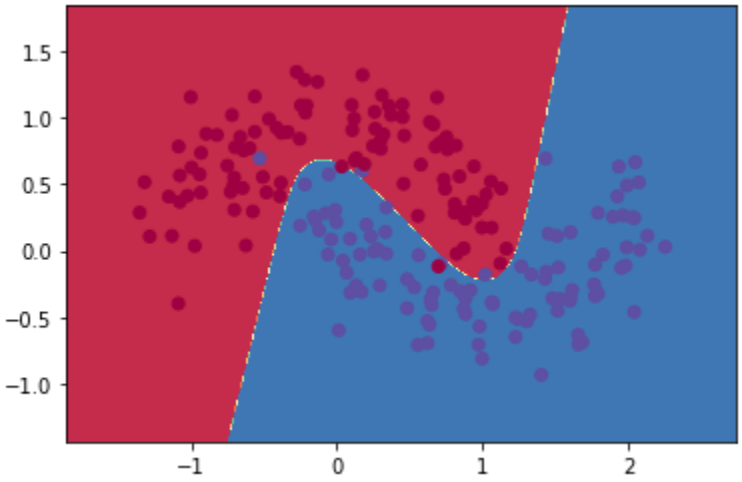
$$\frac{\partial L}{\partial W_1} = X^T \delta_2$$

$$\frac{\partial L}{\partial b_2} = \sum \delta_2$$

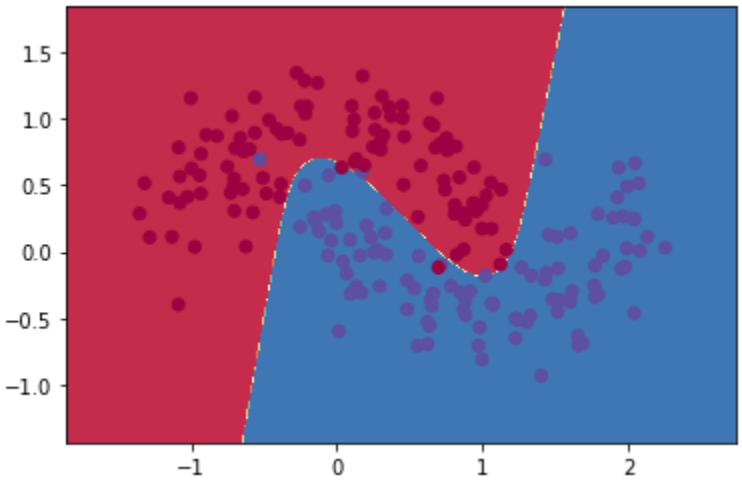
e) Training

a. Decision boundary of training with different activation function

1) Tanh

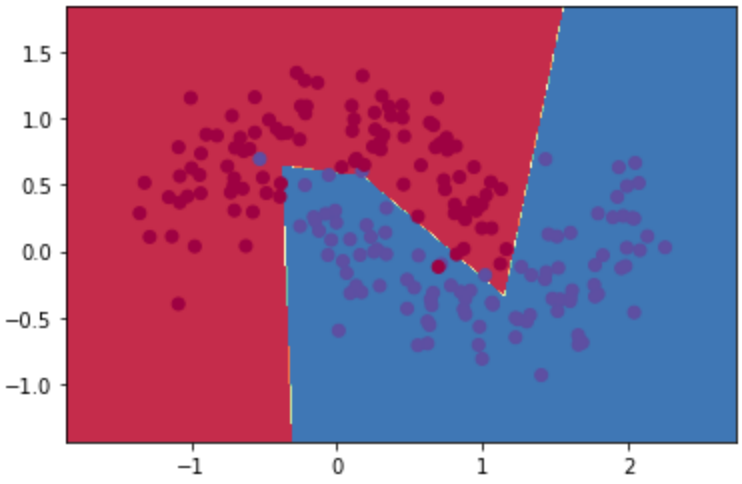


2) Sigmoid



3) Relu

4)

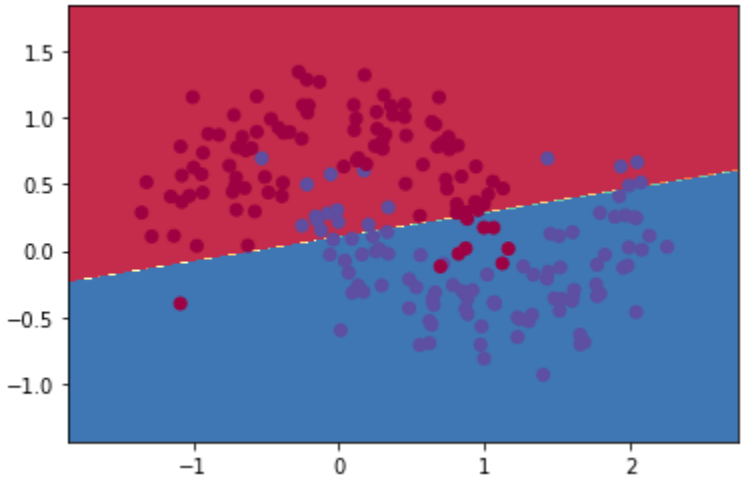


Observations:

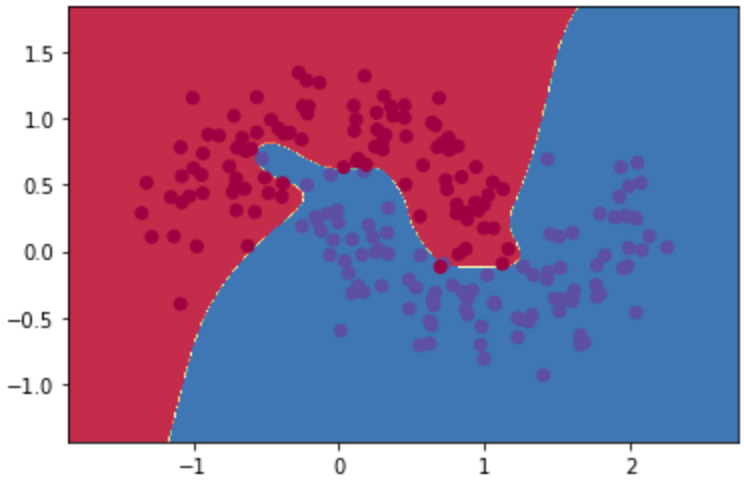
The decision boundary of the relu function is more accurate and the loss is less than the other two functions, and the other functions perform very similar, since the gradient of tanh and sigmoid function are close to 0 when z is very large or small.

b. Different hidden dims

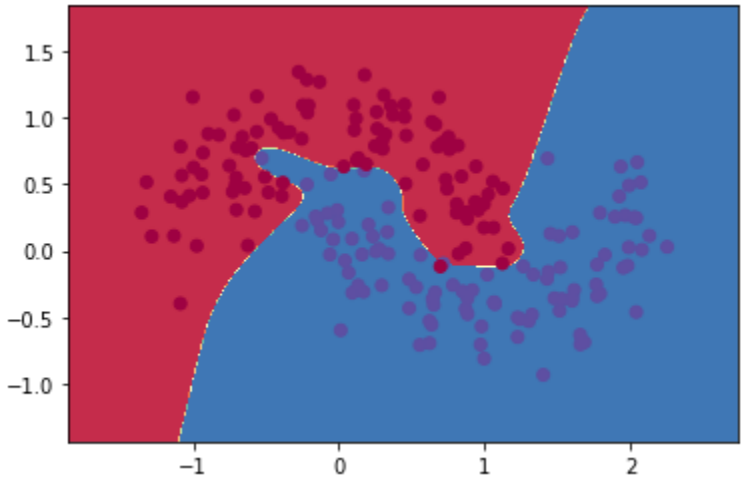
1. 1 dimension



2. 10 dimensions



3. 50 dimensions



Observations:

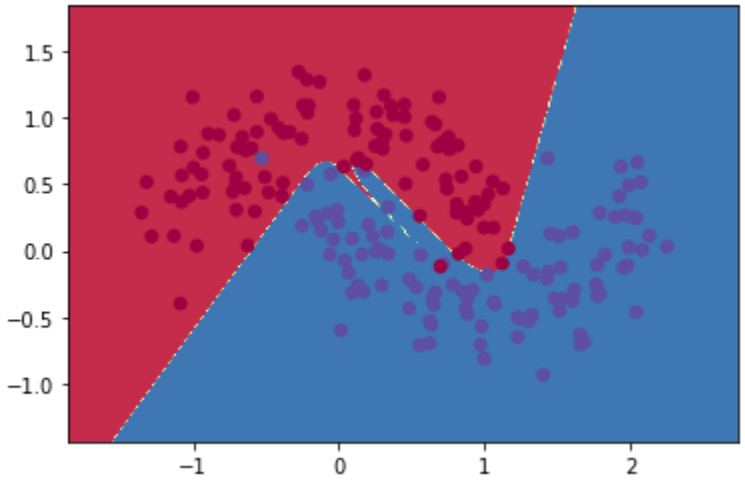
When the dimension is 1, the model cannot fit the data well ,but when the dimensions increase, the model fits the data better than before.
But when the dimension is too big, the training accuracy will be higher than the small one, but I think it will have higher risk to overfit the model.

Task f

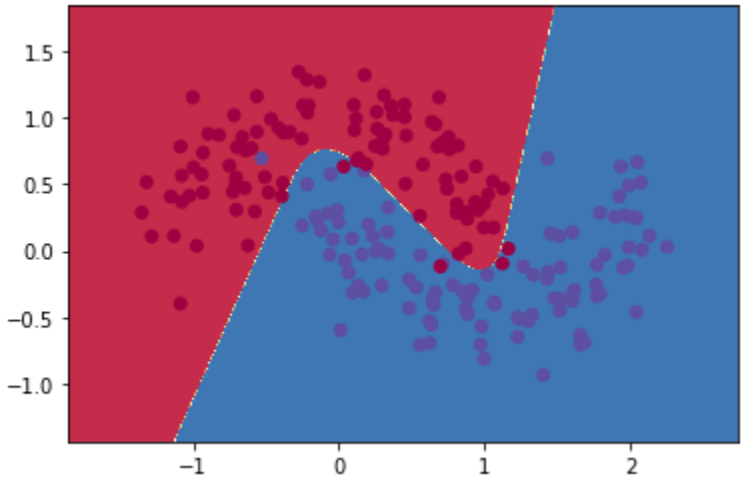
Code
In my code, we decide not to inherit the Neural network class, but to rebuild a new neural network with a list a layer size as the parameters

Please check the n_layer_nerual_network file

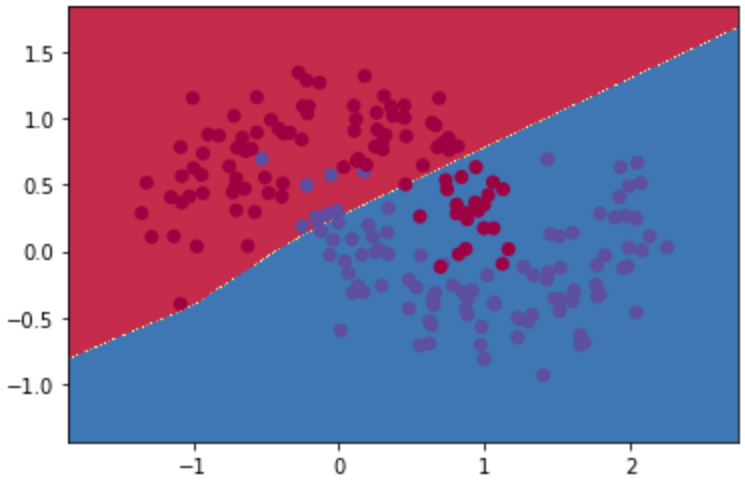
1. Number of hidden layers with the layer size 3
    a. 2 layers



    b. 4 layers

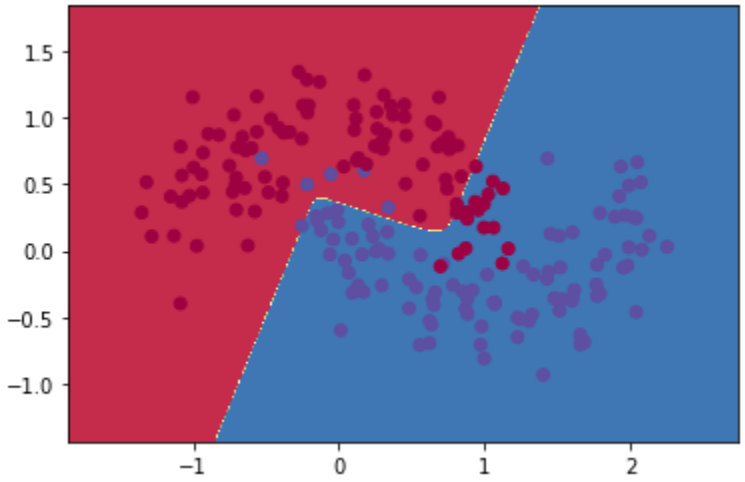


    c. 6 layers

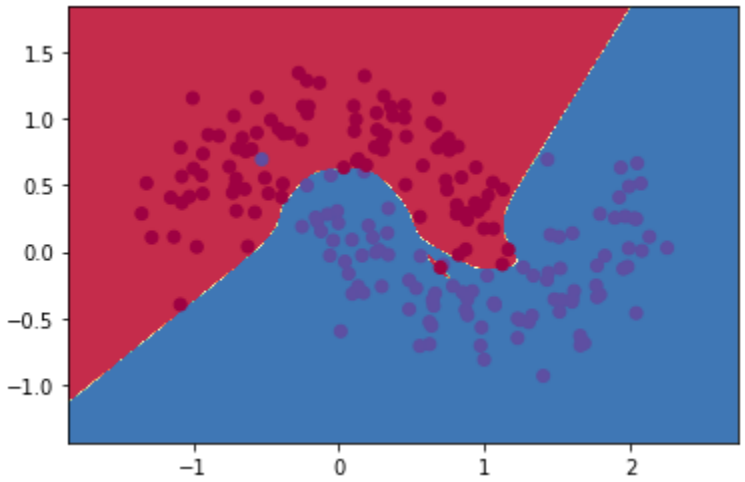


Observations :
When the hidden layers are under 4, the model performs well, but as the hidden layers get bigger, the training accuracy decreases.

2. Layer sizes
    a. Hidden layers size : 2,2

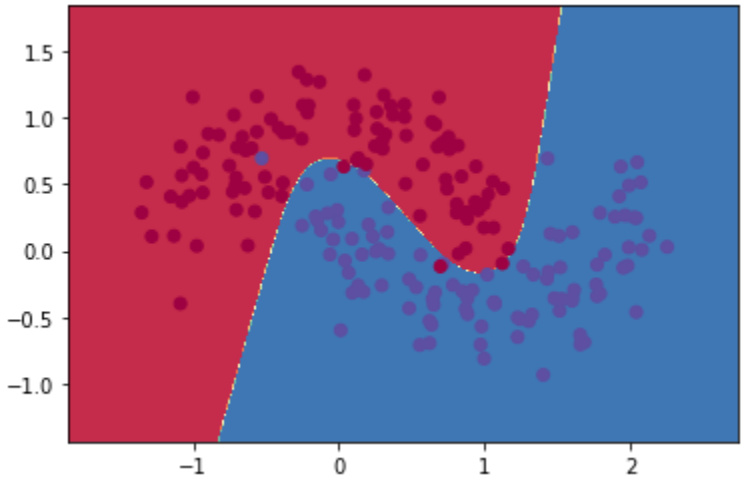


b. Hidden layers size : 4,4



When adding more dims in the neural network, the model performs better than before

3. Active functions
    a. Sigmoid with 2 hidden layers with size 3

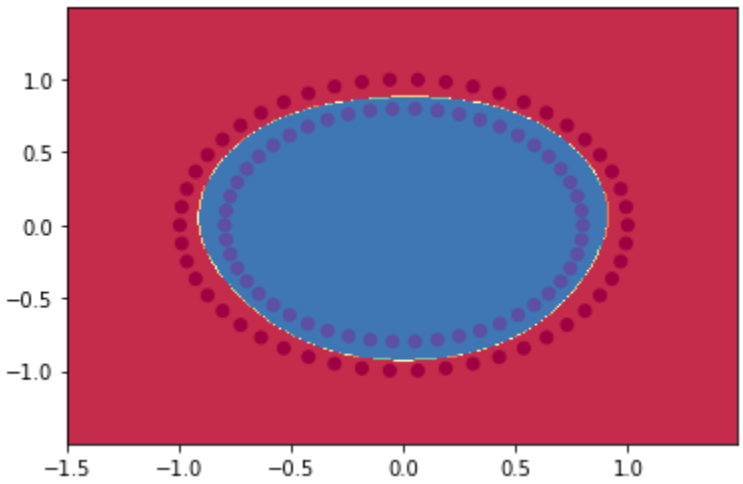


Other datasets

I select the **make_circles** as my dataset, since these two datasets are similar with the dimension and very useful for visualization

```
def generate():
  np.random.seed(0)
  X , y = datasets.make_circles(n_samples=100, shuffle=True, noise=None,
random_state=None, factor=0.8)
  return X,y
```

And the results of the training shows that our model fits it very well



Task 2

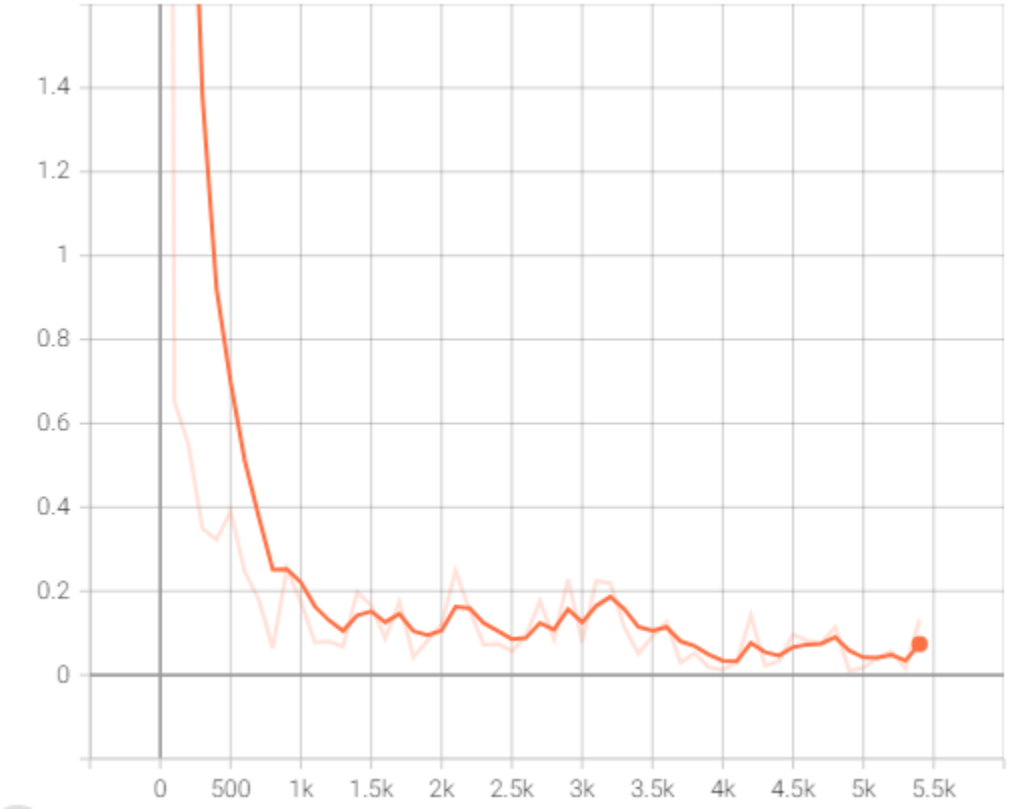
The output results of training are :

step 0, training accuracy 0.16
step 100, training accuracy 0.78
step 200, training accuracy 0.9
step 300, training accuracy 0.98
step 400, training accuracy 0.96
step 500, training accuracy 0.9
step 600, training accuracy 0.96
step 700, training accuracy 0.96
step 800, training accuracy 0.92
step 900, training accuracy 0.94
step 1000, training accuracy 0.96
step 1100, training accuracy 0.98
step 1200, training accuracy 0.94
step 1300, training accuracy 0.96
step 1400, training accuracy 0.96
step 1500, training accuracy 1
step 1600, training accuracy 0.96
step 1700, training accuracy 0.98
step 1800, training accuracy 0.98
step 1900, training accuracy 0.98
step 2000, training accuracy 0.94
step 2100, training accuracy 0.96
step 2200, training accuracy 0.96
step 2300, training accuracy 0.98
step 2400, training accuracy 0.96
step 2500, training accuracy 0.98
step 2600, training accuracy 0.98
step 2700, training accuracy 0.98
step 2800, training accuracy 0.98
step 2900, training accuracy 0.94
step 3000, training accuracy 0.98
step 3100, training accuracy 0.98
step 3200, training accuracy 0.98
step 3300, training accuracy 0.98
step 3400, training accuracy 1
step 3500, training accuracy 1
step 3600, training accuracy 1
step 3700, training accuracy 0.98
step 3800, training accuracy 0.98

step 3900, training accuracy 1
step 4000, training accuracy 1
step 4100, training accuracy 0.98
step 4200, training accuracy 0.98
step 4300, training accuracy 0.98
step 4400, training accuracy 1
step 4500, training accuracy 1
step 4600, training accuracy 1
step 4700, training accuracy 1
step 4800, training accuracy 0.96
step 4900, training accuracy 0.94
step 5000, training accuracy 1
step 5100, training accuracy 1
step 5200, training accuracy 1
step 5300, training accuracy 0.98
step 5400, training accuracy 1
test accuracy 0.9852
The training takes 536.497241 second to finish

Training Loss

1. Figures of scala:



Other parameters

For other parameters such as weights, biases and net input for each layer
The following function can help to summarize them:

```
def variable_summaries(var):
  with tf.name_scope('summaries'):
    mean = tf.reduce_mean(var)
    tf.summary.scalar('mean', mean)
    with tf.name_scope('stddev'):
      stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean))) # standard deviation
    tf.summary.scalar('stddev', stddev)
    tf.summary.scalar('max', tf.reduce_max(var))
    tf.summary.scalar('min', tf.reduce_min(var))
    tf.summary.histogram('histogram', var)
```

And run the code for each layer to generate the figures,

```
variable_summaries(W_conv1)
variable_summaries(b_conv1)
variable_summaries(h_conv1)
variable_summaries(h_pool1)
```

And for the first convolutional layer, the results are the followings:

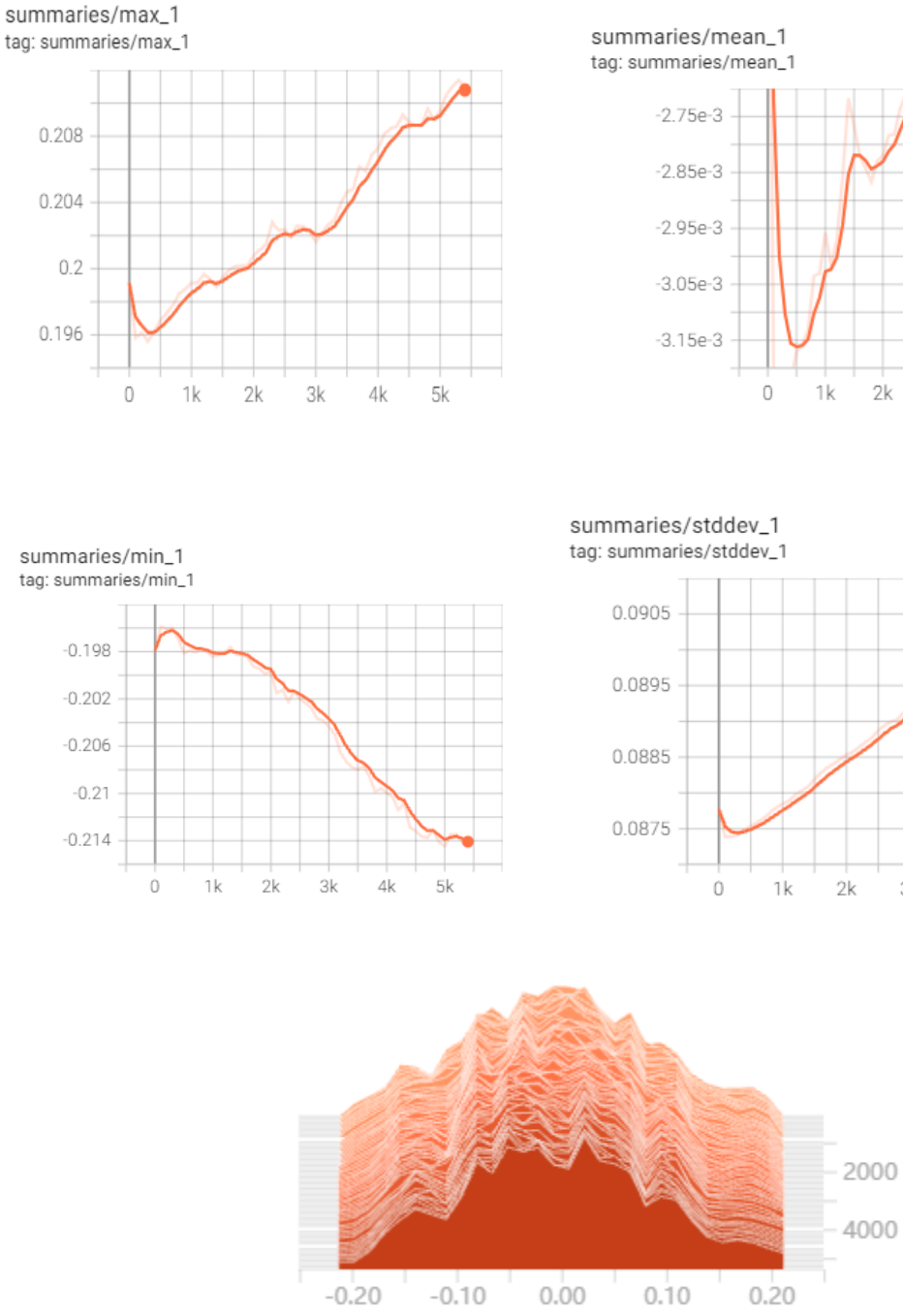
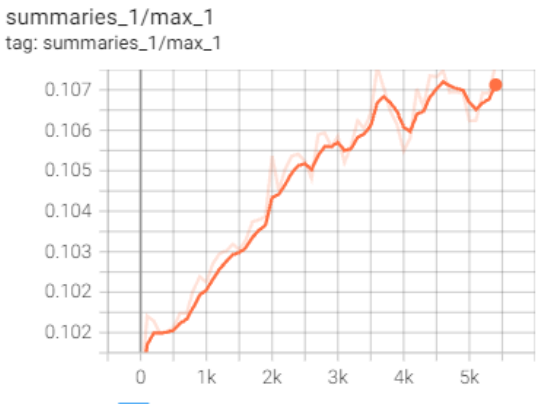
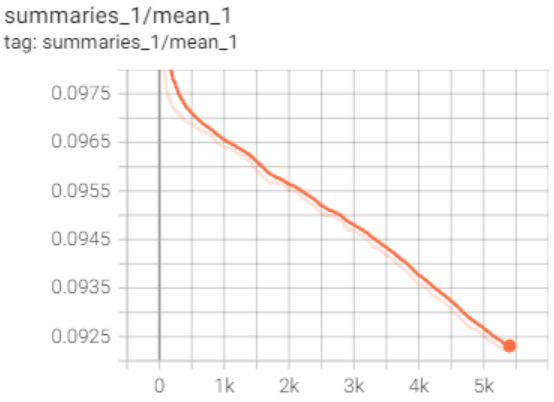
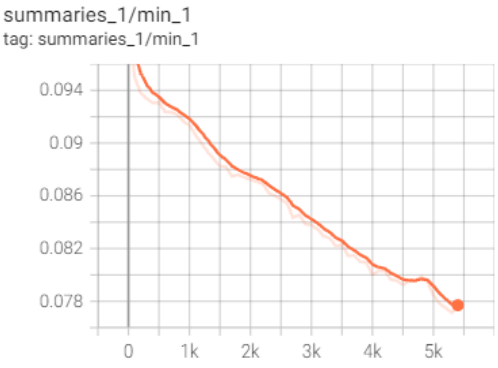
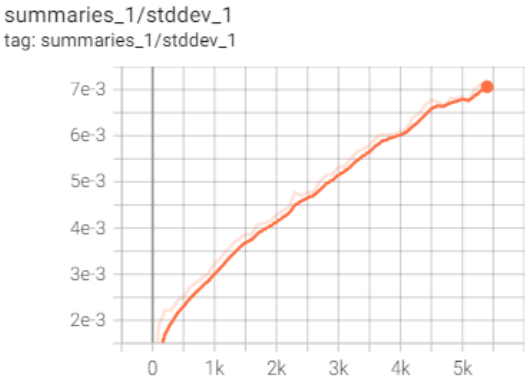


Figure W_conv1









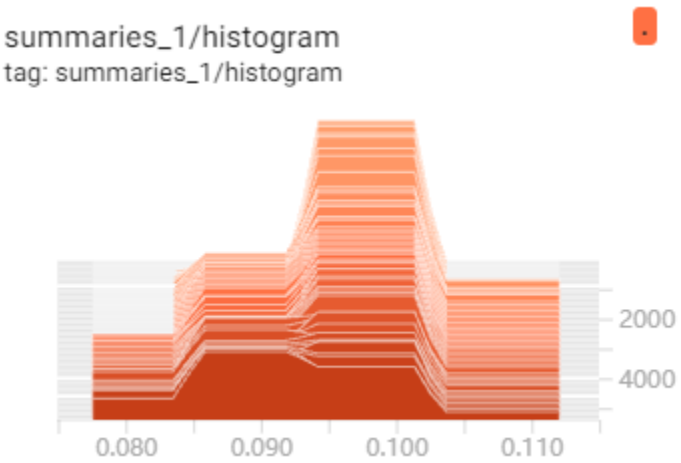


Figure b_conv1

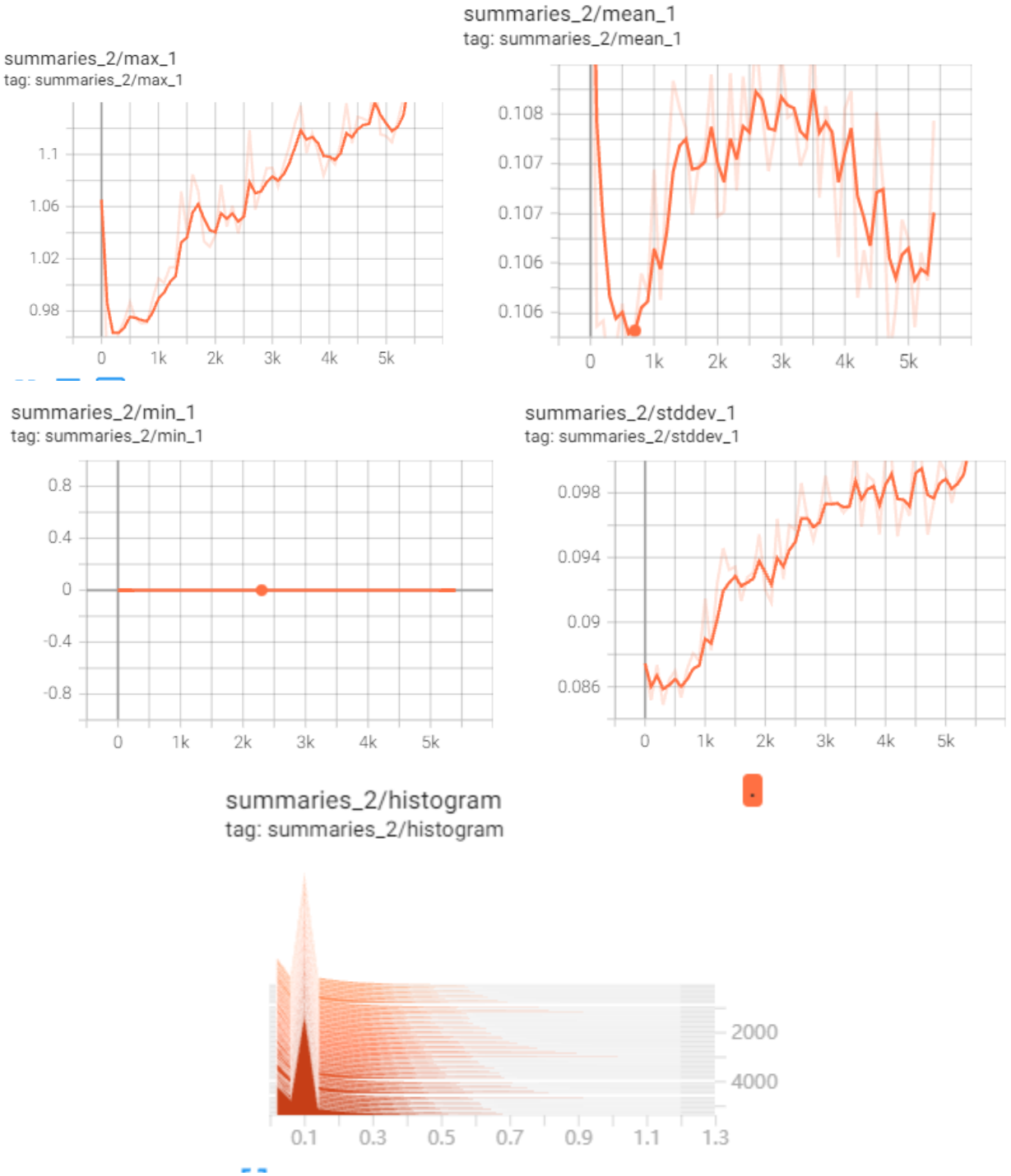


Figure h_conv1

Figure h_pool1

Parameters with different non-linearities and training algo

1. **relu→ tanh**

   ```
   # h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
   h_conv1 = tf.nn.tanh(conv2d(x_image, W_conv1) + b_conv1)
   ```

2. **Adam→ class AdagradDAOptimizer**

   ```
   # train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
   train_step = tf.train.tf.train.GradientDescentOptimizer(1e-4).minimize(cross_entropy)
   ```

the results of training are followings:

step 0, training accuracy 0.04
step 100, training accuracy 0.14
step 200, training accuracy 0.3
step 300, training accuracy 0.28
step 400, training accuracy 0.2
step 500, training accuracy 0.42
step 600, training accuracy 0.32
step 700, training accuracy 0.3
step 800, training accuracy 0.34
step 900, training accuracy 0.52
step 1000, training accuracy 0.54
step 1100, training accuracy 0.6
step 1200, training accuracy 0.6
step 1300, training accuracy 0.64
step 1400, training accuracy 0.68
step 1500, training accuracy 0.52
step 1600, training accuracy 0.52
step 1700, training accuracy 0.7
step 1800, training accuracy 0.62
step 1900, training accuracy 0.68
step 2000, training accuracy 0.8
step 2100, training accuracy 0.7
step 2200, training accuracy 0.66
step 2300, training accuracy 0.82
step 2400, training accuracy 0.76
step 2500, training accuracy 0.78
step 2600, training accuracy 0.72
step 2700, training accuracy 0.74
step 2800, training accuracy 0.68

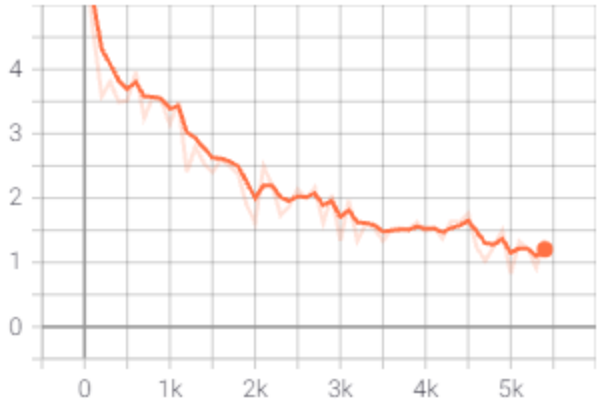
step 2900, training accuracy 0.7
step 3000, training accuracy 0.84
step 3100, training accuracy 0.68
step 3200, training accuracy 0.86
step 3300, training accuracy 0.82
step 3400, training accuracy 0.72
step 3500, training accuracy 0.76
step 3600, training accuracy 0.82
step 3700, training accuracy 0.86
step 3800, training accuracy 0.8
step 3900, training accuracy 0.76
step 4000, training accuracy 0.86
step 4100, training accuracy 0.88
step 4200, training accuracy 0.84
step 4300, training accuracy 0.86
step 4400, training accuracy 0.86
step 4500, training accuracy 0.82
step 4600, training accuracy 0.88
step 4700, training accuracy 0.88
step 4800, training accuracy 0.82
step 4900, training accuracy 0.9
step 5000, training accuracy 0.92
step 5100, training accuracy 0.76
step 5200, training accuracy 0.86
step 5300, training accuracy 0.9
step 5400, training accuracy 0.82

test accuracy 0.8499
The training takes 498.671401 second to finish

Notice that the test accuracy is lower than the last model, and the statistics of each layer is as following:

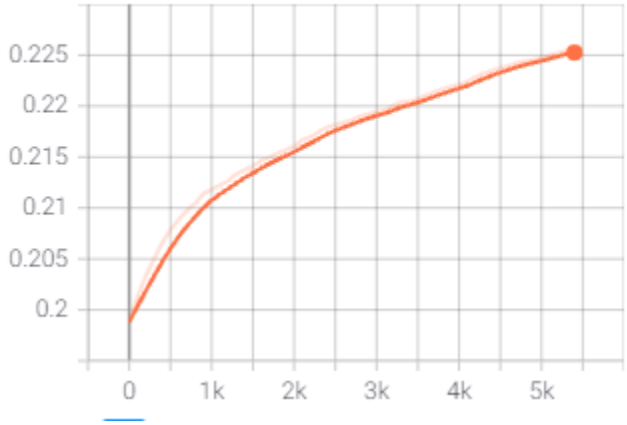
1. Training loss

Mean_1

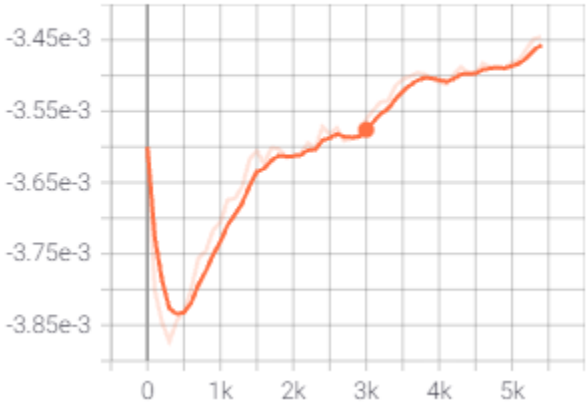


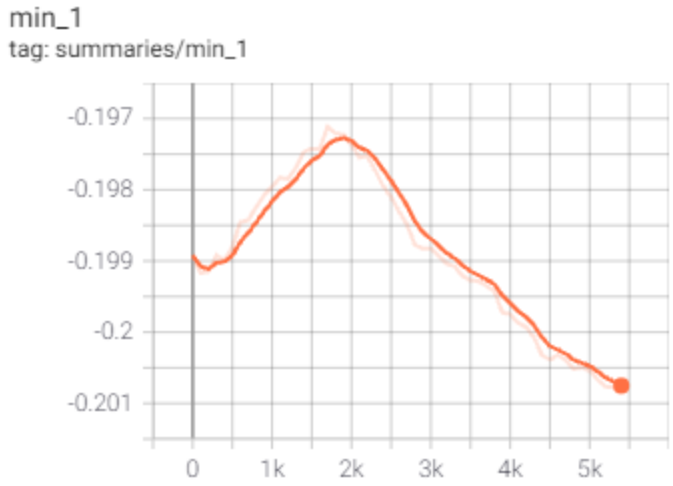
2. Statistics

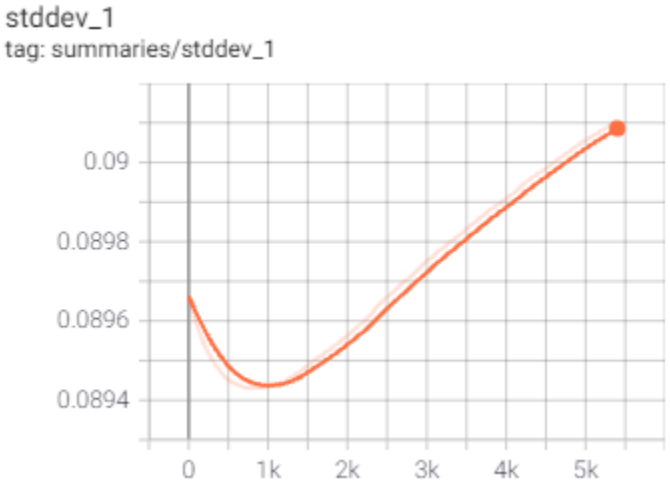
max_1
tag: summaries/max_1



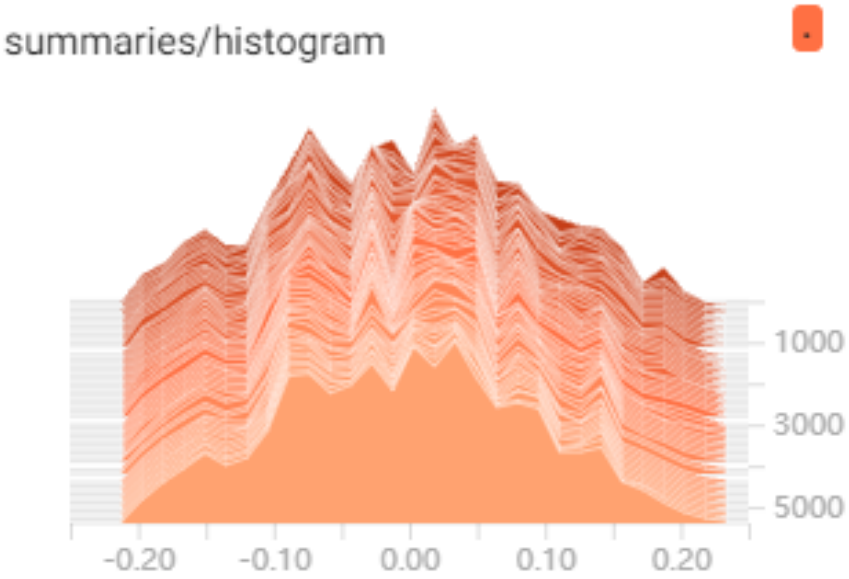
mean_1
tag: summaries/mean_1




min_1
tag: summaries/min_1
-0.197
-0.198
-0.199
-0.2
-0.201
0
1k
2k
3k
4k
5k


stddev_1
tag: summaries/stddev_1
0.09
0.0898
0.0896
0.0894
0
1k
2k
3k
4k
5k

3. Histogram


summaries/histogram
1000
3000
5000
-0.20
-0.10
0.00
0.10
0.20

In general, this model does not perform well on this dataset since both the training loss and test loss are bigger than the old one, and the standard deviation goes bigger as the training process goes on.