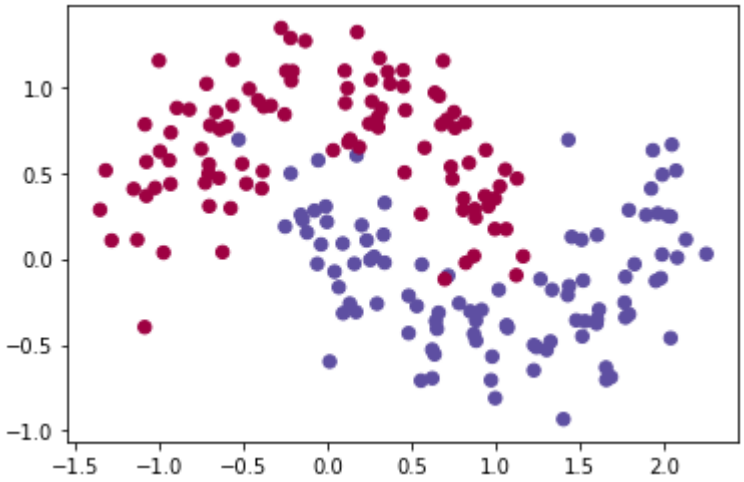
1 Backpropagation in a simple neural network

a) Dataset

Visualization of dataset.

In [5]:
```
from three_layer_neural_network import *
```

In [6]:
```
X, y = generate_data()
plt.scatter(X[:, 0], X[:, 1], s=40, c=y, cmap=plt.cm.Spectral)
plt.show()
```

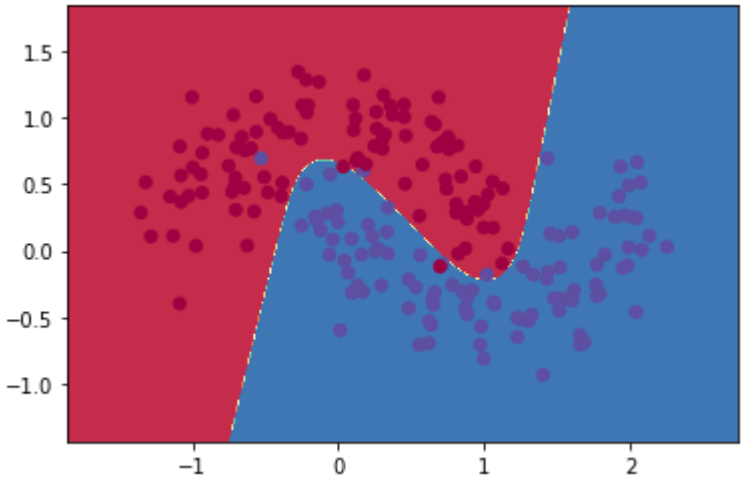


e) Train network with different activation functions

When trying different activation functions, I noticed that ReLU would create a decision boundary with sharp turns while tanh and sigmoid would generate smooth and very similar decision boundaries.

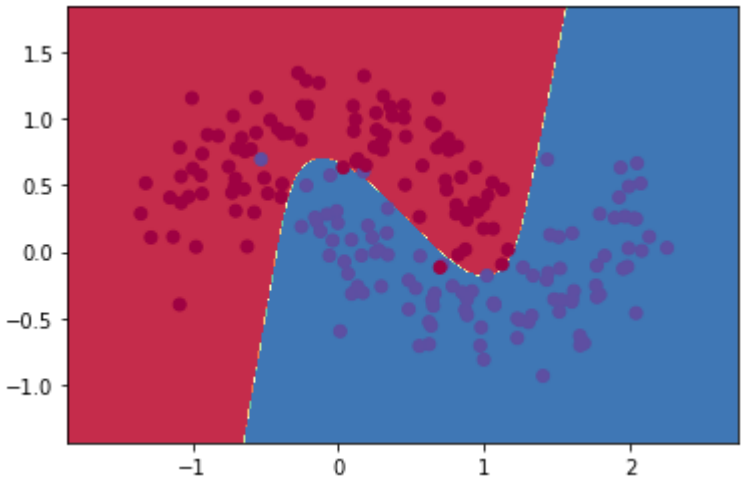
Tanh

In [5]:
```
model = NeuralNetwork(nn_input_dim=2, nn_hidden_dim=3 , nn_output_dim=2, actFu
n_type='tanh')
model.fit_model(X,y,print_loss=False)
model.visualize_decision_boundary(X,y)
```



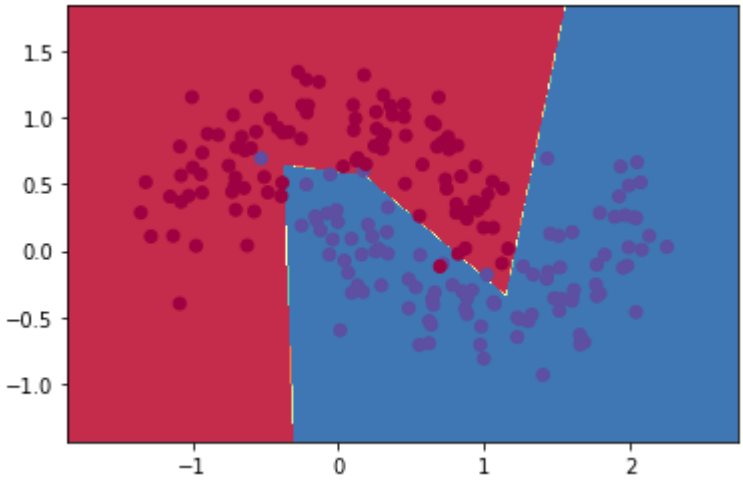
Sigmoid

In [6]:
```
model = NeuralNetwork(nn_input_dim=2, nn_hidden_dim=3 , nn_output_dim=2, actFu
n_type='sigmoid')
model.fit_model(X,y,print_loss=False)
model.visualize_decision_boundary(X,y)
```



ReLU

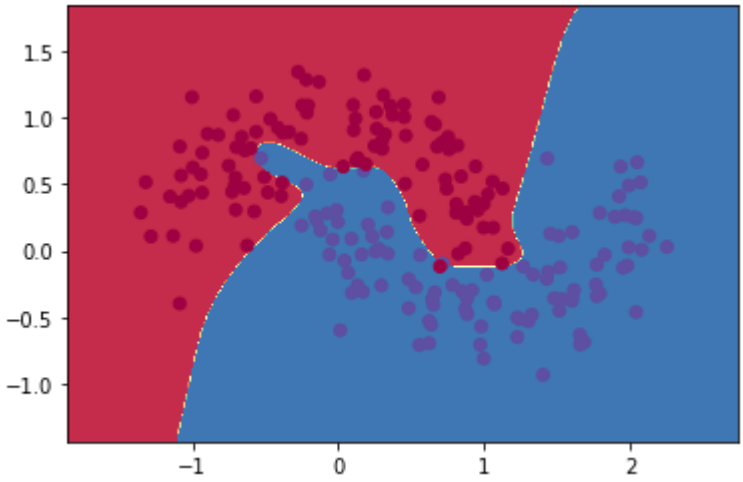
In [7]:
```
model = NeuralNetwork(nn_input_dim=2, nn_hidden_dim=3 , nn_output_dim=2, actFu
n_type='relu')
model.fit_model(X,y,print_loss=False)
model.visualize_decision_boundary(X,y)
```



More hidden units with Tanh

When adding more hidden units, the model starts to overfit the dataset.

In [8]:
```
model = NeuralNetwork(nn_input_dim=2, nn_hidden_dim=20 , nn_output_dim=2, actF
un_type='tanh')
model.fit_model(X,y,print_loss=False)
model.visualize_decision_boundary(X,y)
```



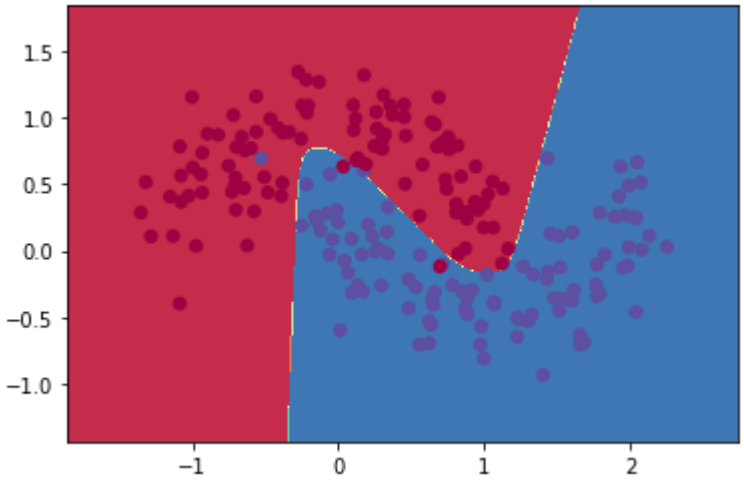
f) Train a Deep Network

When training deeper networks, I noticed their learnability is worse than the three layer network and sometimes the gradients might vanish or explode.

In [13]:
```
from n_layer_neural_network import *
```

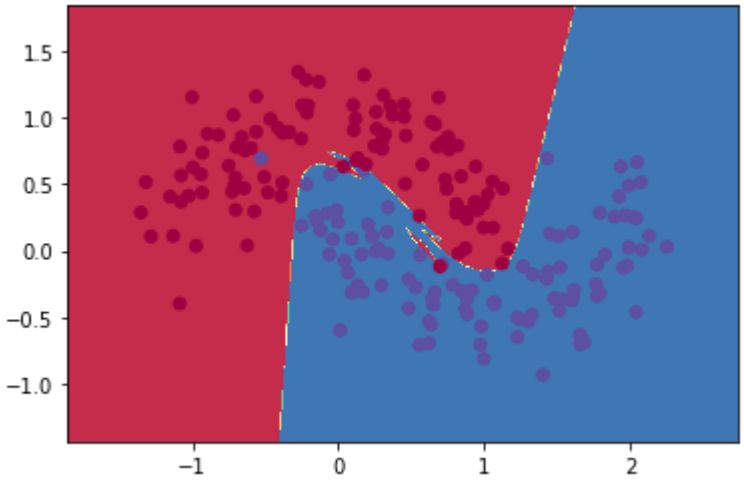
Firstly, I build a model similar to the three layer model with the `n_layer_neural_network` class to verify it. The model performed similarly to the one built with `three_layer_neural_network` .

In [7]:
```
model = DeepNeuralNetwrok(n_hidden=1, input_dim=2, hidden_dim=3, output_dim=2,
actFun_type='tanh')
model.fit_model(X,y,print_loss=False)
model.visualize_decision_boundary(X,y)
```



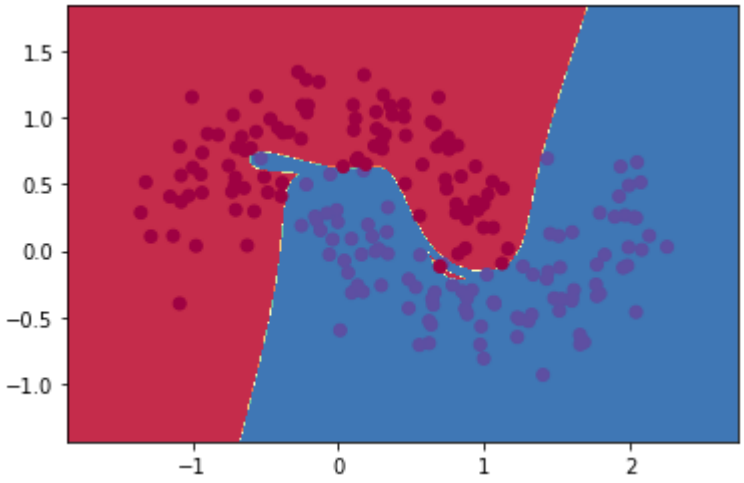
I then build a model with 5 hidden layers and 3 hidden neurons in each layer. The model starts to overfit the dataset and generated some weird turns in the decision boundary.

In [8]:
```
model = DeepNeuralNetwrok(n_hidden=5, input_dim=2, hidden_dim=3, output_dim=2,
actFun_type='tanh')
model.fit_model(X,y,epsilon=0.001,print_loss=False)
model.visualize_decision_boundary(X,y)
```



After adding even more hidden neurons (10 neurons per layer) to the hidden layer, the model reached almost perfect performance by overfitting the dataset. It even created a hole in the decision boudary to exclude the outliers in the dataset. However, the dicision boudary do not reflect the true data structure anymore.

In [9]:
```
model = DeepNeuralNetwrok(n_hidden=5, input_dim=2, hidden_dim=10, output_dim=2
, actFun_type='tanh')
model.fit_model(X,y,epsilon=0.001,print_loss=False)
model.visualize_decision_boundary(X,y)
```

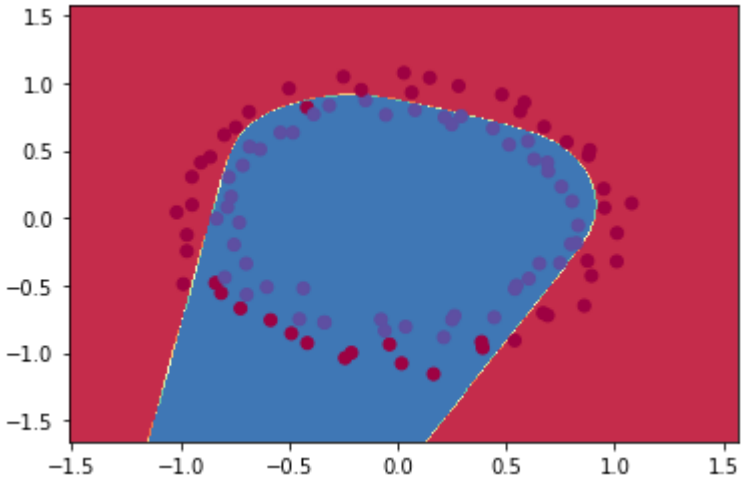


I then tested the deeper neuron network on a more nonlinear dataset to see if the additional layers could help it perform better in those situations. I chose the circle dataset because it requires a more non-linear disentaglement of the data.

In [10]:
```
from sklearn import datasets
np.random.seed(0)
X, y = datasets.make_circles(100, noise=0.05)
```

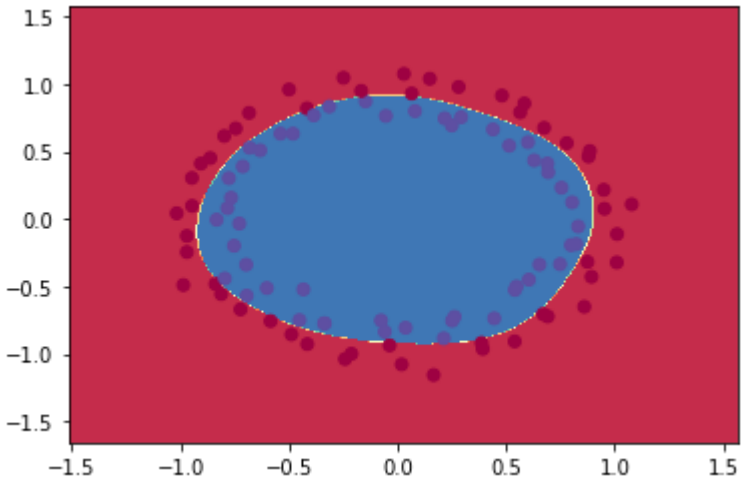
A three layer network could not perform well on this dataset.

In [11]:
```
model = DeepNeuralNetwrok(n_hidden=1, input_dim=2, hidden_dim=3, output_dim=2,
actFun_type='tanh')
model.fit_model(X,y,print_loss=False)
model.visualize_decision_boundary(X,y)
```



A deep network with 5 hidden layers and 10 neurons in each layer performed almost perfectly on this dataset, indicating that deeper neural networks could perform more non-linear tasks than shallow ones.

In [12]:
```
model = DeepNeuralNetwrok(n_hidden=5, input_dim=2, hidden_dim=10, output_dim=2
, actFun_type='tanh')
model.fit_model(X,y,epsilon=0.001,print_loss=False)
model.visualize_decision_boundary(X,y)
```



2 Training a Simple Deep Convolutional Network on MNIST

a) Build and Train a 4-layer DCN

In [1]:
```
from dcn_mnist import *
```

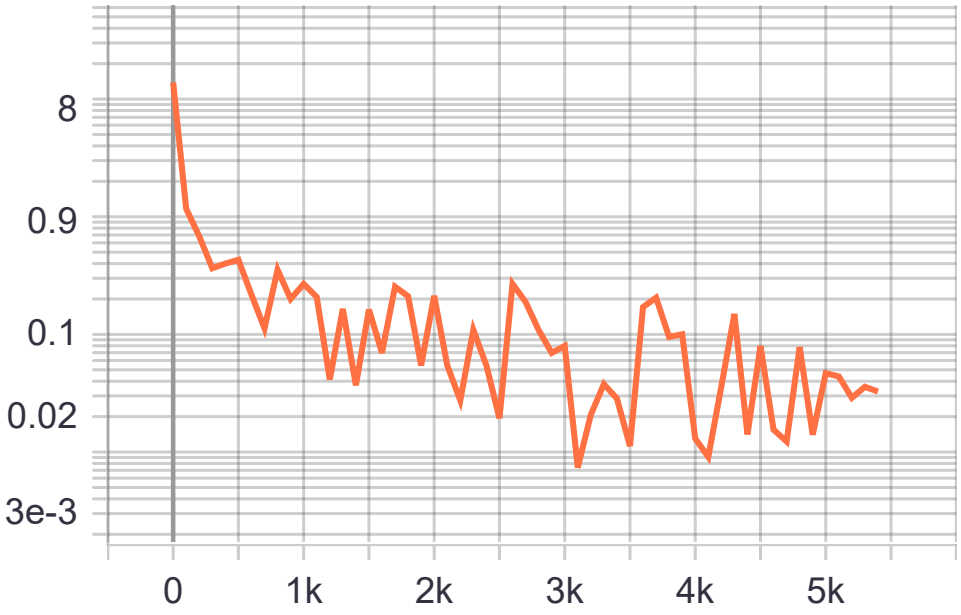
```
WARNING: Logging before flag parsing goes to stderr.
W1016 01:28:15.933898 140587956586304 deprecation.py:323] From /COMP576/dcn_m
nist.py:6: read_data_sets (from tensorflow.contrib.learn.python.learn.dataset
s.mnist) is deprecated and will be removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/mod
els.
W1016 01:28:15.934455 140587956586304 deprecation.py:323] From /usr/local/li
b/python3.6/dist-packages/tensorflow/contrib/learn/python/learn/datasets/mnis
t.py:260: maybe_download (from tensorflow.contrib.learn.python.learn.dataset
s.base) is deprecated and will be removed in a future version.
Instructions for updating:
Please write your own downloading logic.
W1016 01:28:15.934968 140587956586304 deprecation.py:323] From /usr/local/li
b/python3.6/dist-packages/tensorflow/contrib/learn/python/learn/datasets/mnis
t.py:262: extract_images (from tensorflow.contrib.learn.python.learn.dataset
s.mnist) is deprecated and will be removed in a future version.
Instructions for updating:
Please use tf.data to implement this functionality.
W1016 01:28:16.076868 140587956586304 deprecation.py:323] From /usr/local/li
b/python3.6/dist-packages/tensorflow/contrib/learn/python/learn/datasets/mnis
t.py:267: extract_labels (from tensorflow.contrib.learn.python.learn.dataset
s.mnist) is deprecated and will be removed in a future version.
Instructions for updating:
Please use tf.data to implement this functionality.
W1016 01:28:16.078018 140587956586304 deprecation.py:323] From /usr/local/li
b/python3.6/dist-packages/tensorflow/contrib/learn/python/learn/datasets/mnis
t.py:110: dense_to_one_hot (from tensorflow.contrib.learn.python.learn.datase
ts.mnist) is deprecated and will be removed in a future version.
Instructions for updating:
Please use tf.one_hot on tensors.
W1016 01:28:16.104194 140587956586304 deprecation.py:323] From /usr/local/li
b/python3.6/dist-packages/tensorflow/contrib/learn/python/learn/datasets/mnis
t.py:290: DataSet.__init__ (from tensorflow.contrib.learn.python.learn.datase
ts.mnist) is deprecated and will be removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/mod
els.

Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz

W1016 01:28:16.214524 140587956586304 deprecation_wrapper.py:119] From /COMP5
76/dcn_mnist.py:10: The name tf.InteractiveSession is deprecated. Please use
tf.compat.v1.InteractiveSession instead.
```

In [2]:
```
main()
```

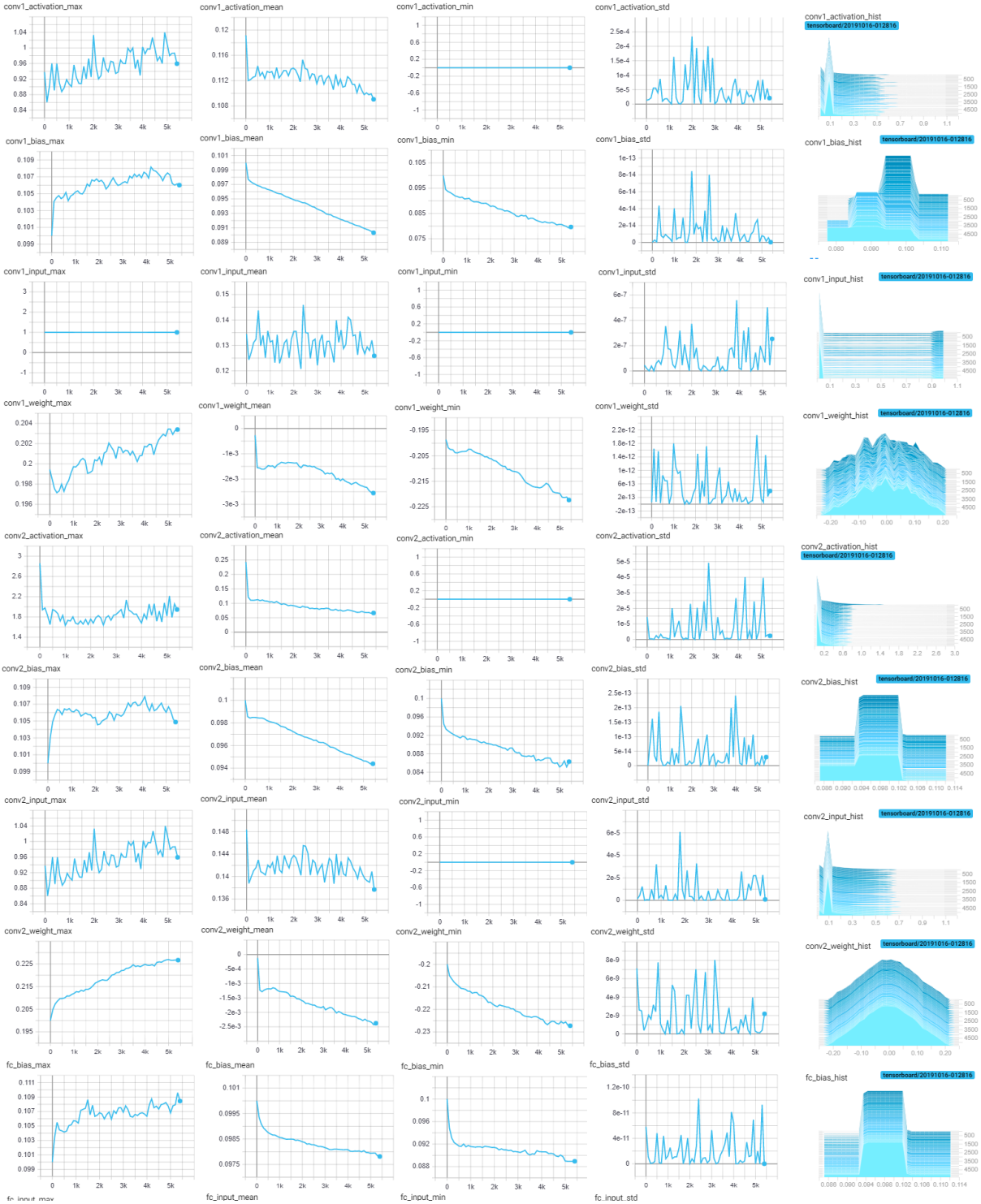
```
W1016 01:28:16.328822 140587956586304 deprecation_wrapper.py:119] From /COMP5
76/dcn_mnist.py:81: The name tf.placeholder is deprecated. Please use tf.comp
at.v1.placeholder instead.

W1016 01:28:16.331760 140587956586304 deprecation_wrapper.py:119] From /COMP5
76/dcn_mnist.py:24: The name tf.truncated_normal is deprecated. Please use t
f.random.truncated_normal instead.

W1016 01:28:16.359403 140587956586304 deprecation.py:506] From /COMP576/dcn_m
nist.py:107: calling dropout (from tensorflow.python.ops.nn_ops) with keep_pr
ob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - k
eep_prob`.
W1016 01:28:16.373851 140587956586304 deprecation_wrapper.py:119] From /COMP5
76/dcn_mnist.py:117: The name tf.log is deprecated. Please use tf.math.log in
stead.

W1016 01:28:16.376964 140587956586304 deprecation_wrapper.py:119] From /COMP5
76/dcn_mnist.py:118: The name tf.train.AdamOptimizer is deprecated. Please us
e tf.compat.v1.train.AdamOptimizer instead.

W1016 01:28:16.556908 140587956586304 deprecation_wrapper.py:119] From /COMP5
76/dcn_mnist.py:132: The name tf.summary.scalar is deprecated. Please use tf.
compat.v1.summary.scalar instead.

W1016 01:28:16.570427 140587956586304 deprecation_wrapper.py:119] From /COMP5
76/dcn_mnist.py:130: The name tf.summary.histogram is deprecated. Please use
tf.compat.v1.summary.histogram instead.

W1016 01:28:16.708732 140587956586304 deprecation_wrapper.py:119] From /COMP5
76/dcn_mnist.py:150: The name tf.summary.merge_all is deprecated. Please use
tf.compat.v1.summary.merge_all instead.

W1016 01:28:16.710836 140587956586304 deprecation_wrapper.py:119] From /COMP5
76/dcn_mnist.py:153: The name tf.summary.merge is deprecated. Please use tf.c
ompat.v1.summary.merge instead.

W1016 01:28:16.713041 140587956586304 deprecation.py:323] From /usr/local/li
b/python3.6/dist-packages/tensorflow/python/util/tf_should_use.py:193: initia
lize_all_variables (from tensorflow.python.ops.variables) is deprecated and w
ill be removed after 2017-03-02.
Instructions for updating:
Use `tf.global_variables_initializer` instead.
W1016 01:28:16.714064 140587956586304 deprecation_wrapper.py:119] From /COMP5
76/dcn_mnist.py:159: The name tf.train.Saver is deprecated. Please use tf.com
pat.v1.train.Saver instead.

W1016 01:28:16.731873 140587956586304 deprecation_wrapper.py:119] From /COMP5
76/dcn_mnist.py:162: The name tf.summary.FileWriter is deprecated. Please use
tf.compat.v1.summary.FileWriter instead.
```

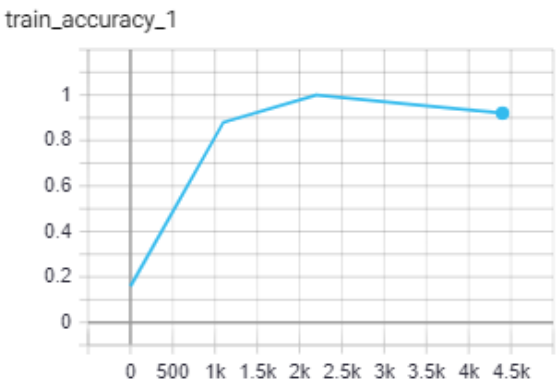
```
step 0, training accuracy 0.1
step 100, training accuracy 0.8
step 200, training accuracy 0.88
step 300, training accuracy 0.92
step 400, training accuracy 0.96
step 500, training accuracy 0.92
step 600, training accuracy 0.92
step 700, training accuracy 0.96
step 800, training accuracy 0.92
step 900, training accuracy 0.96
step 1000, training accuracy 0.96
step 1100, training accuracy 0.96
step 1200, training accuracy 1
step 1300, training accuracy 1
step 1400, training accuracy 1
step 1500, training accuracy 0.92
step 1600, training accuracy 0.98
step 1700, training accuracy 0.98
step 1800, training accuracy 0.98
step 1900, training accuracy 0.96
step 2000, training accuracy 0.96
step 2100, training accuracy 0.98
step 2200, training accuracy 1
step 2300, training accuracy 0.98
step 2400, training accuracy 1
step 2500, training accuracy 0.98
step 2600, training accuracy 0.98
step 2700, training accuracy 1
step 2800, training accuracy 0.98
step 2900, training accuracy 0.96
step 3000, training accuracy 1
step 3100, training accuracy 0.98
step 3200, training accuracy 1
step 3300, training accuracy 0.98
step 3400, training accuracy 0.94
step 3500, training accuracy 1
step 3600, training accuracy 0.98
step 3700, training accuracy 1
step 3800, training accuracy 0.98
step 3900, training accuracy 0.98
step 4000, training accuracy 0.98
step 4100, training accuracy 1
step 4200, training accuracy 1
step 4300, training accuracy 0.98
step 4400, training accuracy 0.98
step 4500, training accuracy 1
step 4600, training accuracy 1
step 4700, training accuracy 0.98
step 4800, training accuracy 0.98
step 4900, training accuracy 1
step 5000, training accuracy 1
step 5100, training accuracy 0.98
step 5200, training accuracy 1
step 5300, training accuracy 1
step 5400, training accuracy 1
test accuracy 0.9875
The training takes 33.965940 second to finish
```

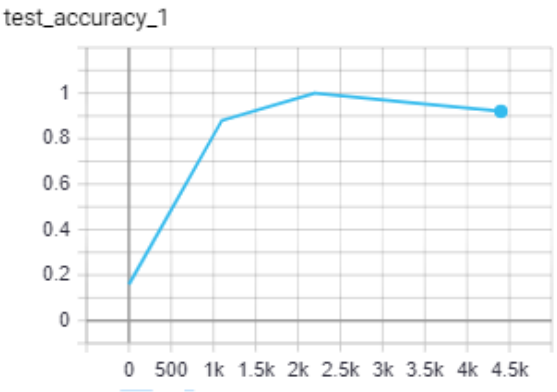
Monitor cross entropy loss through training



b) More on Visualizing Your Training


conv1_activation_max
conv1_activation_mean
conv1_activation_min
conv1_activation_std
conv1_activation_hist
tensorboard/20191016-012816
conv1_bias_max
conv1_bias_mean
conv1_bias_min
conv1_bias_std
conv1_bias_hist
tensorboard/20191016-012816
conv1_input_max
conv1_input_mean
conv1_input_min
conv1_input_std
conv1_input_hist
tensorboard/20191016-012816
conv1_weight_max
conv1_weight_mean
conv1_weight_min
conv1_weight_std
conv1_weight_hist
tensorboard/20191016-012816
conv2_activation_max
conv2_activation_mean
conv2_activation_min
conv2_activation_std
conv2_activation_hist
tensorboard/20191016-012816
conv2_bias_max
conv2_bias_mean
conv2_bias_min
conv2_bias_std
conv2_bias_hist
tensorboard/20191016-012816
conv2_input_max
conv2_input_mean
conv2_input_min
conv2_input_std
conv2_input_hist
tensorboard/20191016-012816
conv2_weight_max
conv2_weight_mean
conv2_weight_min
conv2_weight_std
conv2_weight_hist
tensorboard/20191016-012816
fc_bias_max
fc_bias_mean
fc_bias_min
fc_bias_std
fc_bias_hist
tensorboard/20191016-012816
fc_input_max
fc_input_mean
fc_input_min
fc_input_std



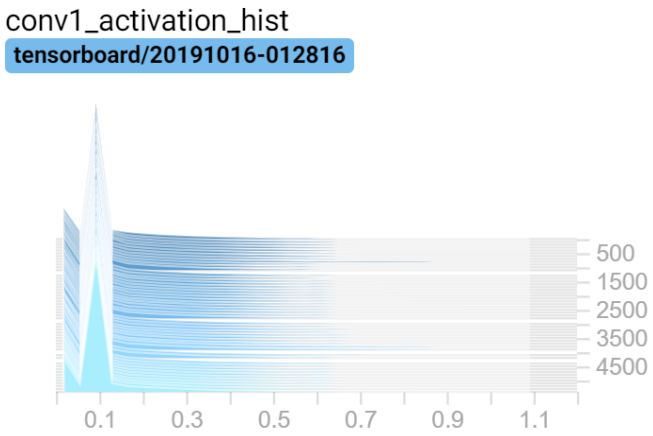


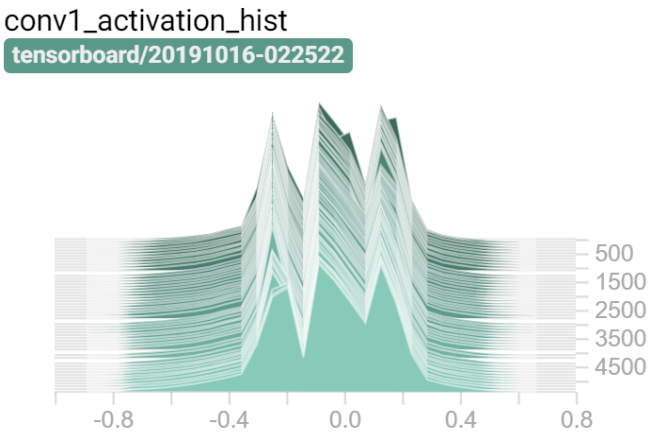
c) Time for More Fun!

In the previous question, we used ReLU for activation and truncated uniform distribution for initialization. From the histogram generated by the tensorboard, we could clearly see that most of the neurons in the network are 'dead neurons' which have zero activation. During backprop, those 'dead neurons' would not let gradient flow and could lead to slower learning. To deal with this, I used tanh instead of ReLU for activation and initialized the weights and biases using Xavier initialization. As expected, the distribution of neuron acivation more resembles normal distribution and most of the neurons in the network have activation. The nerwork also learns slightly faster as indicated in the test accuracy plot.

Histogram of activation in layer conv_1

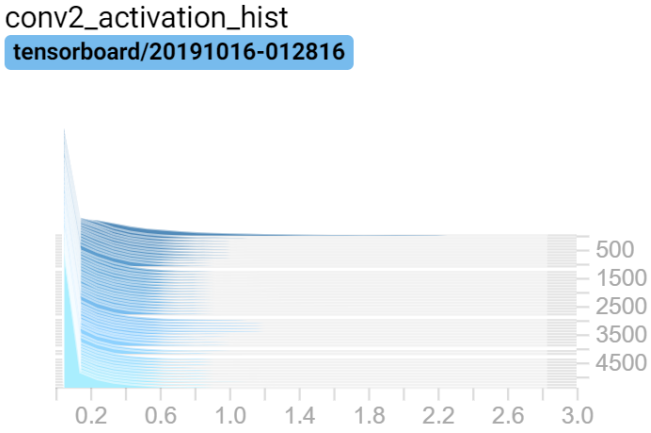
Blue: network trained with ReLU activation and uniform initialization Green: network trained with tanh activation and Xavier initialization

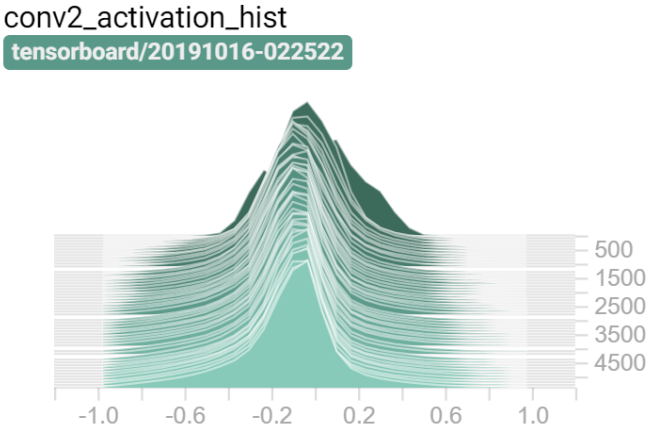




Histogram of activation in layer conv_2

Blue: network trained with ReLU activation and uniform initialization Green: network trained with tanh activation and Xavier initialization





Test accuracy plot

Blue: network trained with ReLU activation and uniform initialization Green: network trained with tanh activation and Xavier initialization