

COMP 576 HW 1

Peter Tang

October 4, 2017

1 1. Backpropagation in a Simple Neural Network

1.1 Dataset

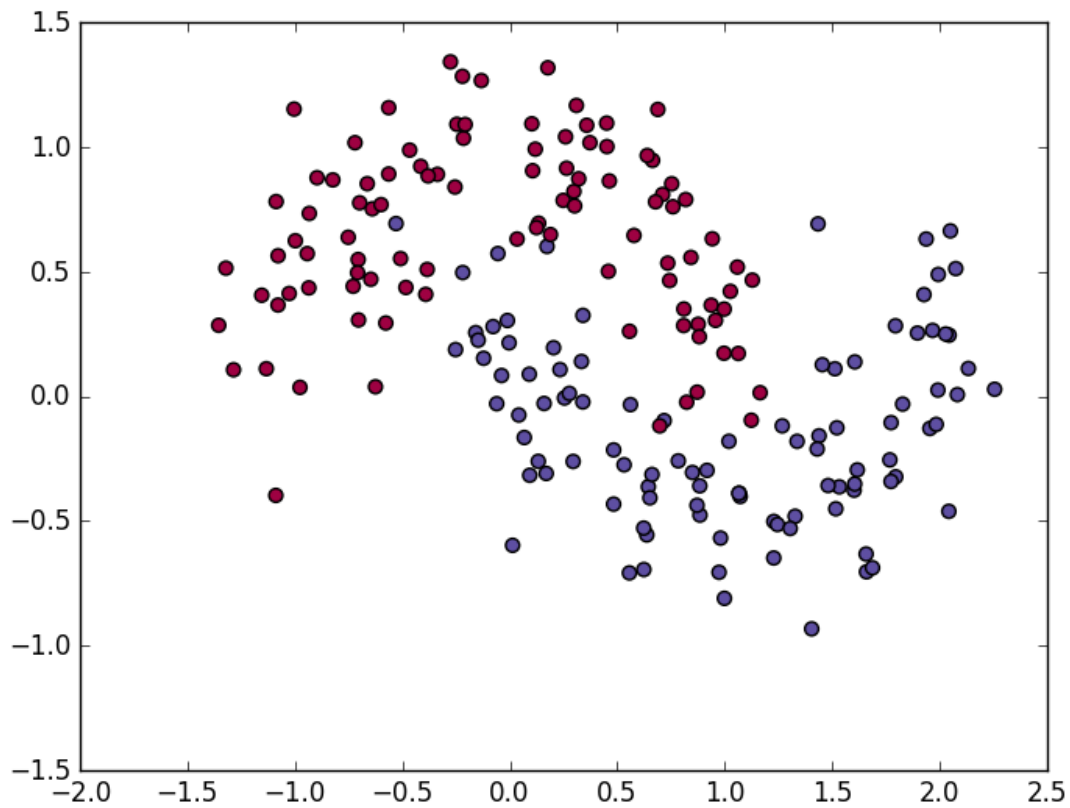


Figure 1: Make-moon Data

1.2 Activation function

$$b) \quad S(x) = \frac{e^x}{e^x + 1} \Rightarrow \frac{ds}{dx} = \frac{(e^x + 1)e^x - e^{2x}}{(e^x + 1)^2}$$

(sigmoid)
$$= \frac{e^x}{(e^x + 1)} \left(\frac{e^x + 1 - e^x}{e^x + 1} \right) = \left(\frac{e^x}{e^x + 1} \right) \left(1 - \frac{e^x}{e^x + 1} \right)$$

$$= S(x) \times (1 - S(x))$$

(tanh)
$$T(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{(e^x + e^{-x})(e^x - e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2}$$

$$= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} = 1 - [\tanh(x)]^2$$

(Relu)
$$R(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

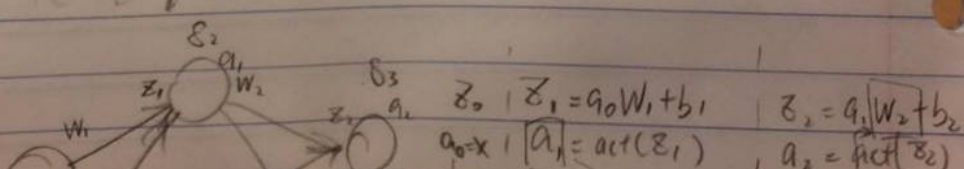


Figure 2: Derivation for derivatives of activation functions

1.3 Build the Neural Network

1.4 Backward Pass – Backpropagation

$$\begin{aligned}
 \frac{\partial L}{\partial w_2} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_2} = \delta_3 a_1 \\
 \frac{\partial L}{\partial b_2} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial b_2} = \delta_2 \\
 \frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} = \delta_2 w_2 \left(\frac{\text{derivative of activation}}{\text{activation}} \right) x = \delta_2 x \\
 \frac{\partial L}{\partial b_1} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1} = \delta_2 w_2 \left(\frac{\text{derivative of activation}}{\text{activation}} \right) = \delta_2
 \end{aligned}$$

Figure 3: Backpropagation Derivation

1.5 Time to Have Fun - Training!

1.5.1 Activations

As shown in 4 Three activation generally learn well with given learning rate and lambda. ReLU activation results in boundary condition that is almost piecewise linear equations. This could be due to the fact that ReLU itself has a linear cut off and kills the signals that are below 0, resulting in blunt edges. The sigmoid and tanh activation functions result in fairly similar boundary conditions. They all look like 3rd degree polynomial functions.

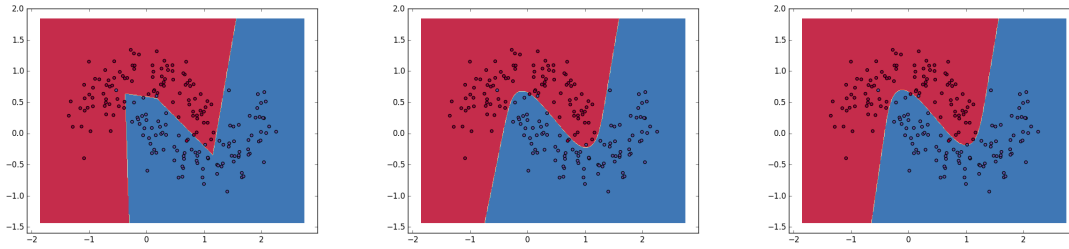


Figure 4: Default setting with different activation function (left: ReLU; middle: Tanh; right: Sigmoid)

1.5.2 Hidden Units

This time, we are varying the number of hidden units within the hidden layer.

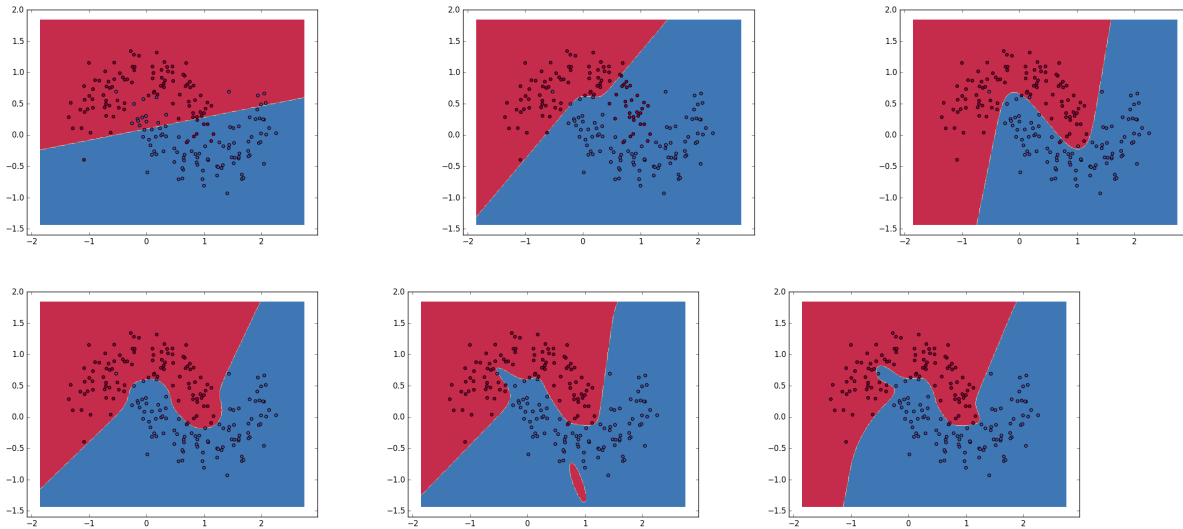


Figure 5: Tanh with different number of hidden units (top row: 3,4,5 hidden units; bottom row: 6,10,20 hidden units)

1.6 Even more fun - training a deeper Network!!!

I built a `DeepNeuralNetwork` class and a `Layer` class. Within `Layer.feedforward` and `Layer.backprop` I implemented normal layer calculations. And within `Layer.init`, I take in number of input and output to randomly initiate the weights. Within `DeepNeuralNetwork`, I initiate the class object by adding layers attribute to the object and it is a list of `Layer` objects.

First I will explore within Sigmoid activation function, and I will try to increase layers with the 3 hidden nodes on each layer, below are the results For sigmoid as activation function (). The deeper the model is, the more peculiar the model are about the lambda and epsilon. At 9 hidden layer depth with 3 nodes on each layer, I cannot manually find a good combination of parameter that allows the model to converge properly. Could be sigmoid function takes longer to converge.

boundary expressiveness are usually set by first few layers. If you cut down dimension in second layer, the information is lost for the later information. If you sift it down slowly, the extra expressive power can then be passed down

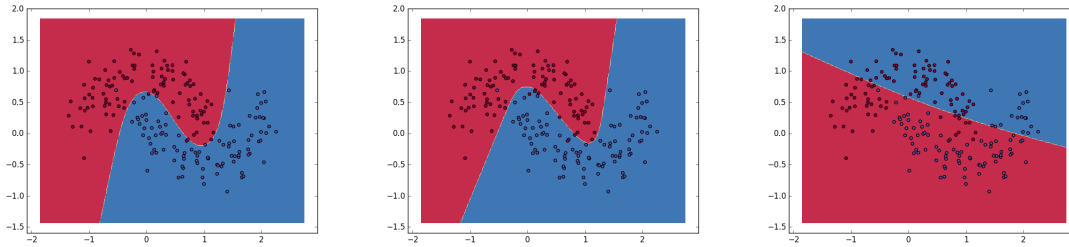


Figure 6: Exploring depth with same layer size in Sigmoid (left: 3 hidden layer; middle: 6 hidden layer; right: 9 hidden layer)

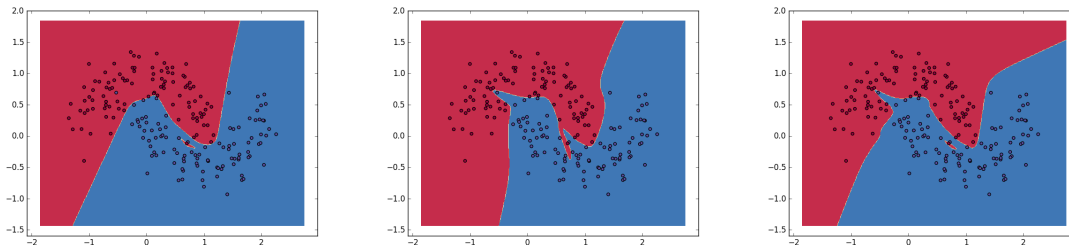


Figure 7: Exploring architecture with tanh (left: hidden layer [6, 20, 6]; middle: hidden layer [20, 20, 6]; right: hidden layer [6, 20, 20])

Tanh boundaries are generally more expressive than those of Sigmoid. The learning rate and lambda also are easier to set without exploding.

Here are some additional models I created. Due to limited parameter in dataset, we aren't able to exploit the expressiveness of the deep neural network. ReLU at with high depth can approximate smooth surface well with piecewise linear boundary.

I also explored training the NN on breast cancer dataset.

Initially I tried the following two models:

1. sigma = 0.01, lambda = 0.0001, layers = [100, 100], tanh
2. sigma = 0.00005, lambda = 0.0001, layers = [30, 30, 30], sigmoid

The second model performed quite well, compared to the first one, however, the computation is extremely slow, I then decide to normalize the data first before building the model

After normalization, the computation is much faster. I was also able to build a few more model that has pretty good looking training curve

3. sigma = 0.000001, lambda = 0.0001, layers = [10, 10, 10, 10, 10], ReLU
4. sigma = 0.00005, lambda = 0.0001, layers = [100], ReLU
5. sigma = 0.0001, lambda = 0.0001, layers = [10, 5], tanh

To my surprise, to capture a relatively higher dimension data (30), small NN of [10,5] could work really well. Normalizing the data also helped a lot with not only loss but also calculation time. Learning rate also needs to be really small in order for deep NN to catch up with deeper parameters. I would assume model 3 would be very good at capturing features with it's deep structure, however, the loss curve seems to indicate that there is not enough capacity for this variability. Comparing with un-normalized data, model 2 is quite a miracle considering how well it has performed relatively. I wonder if it has anything to do with sigmoid function, which usually doesn't perform as well

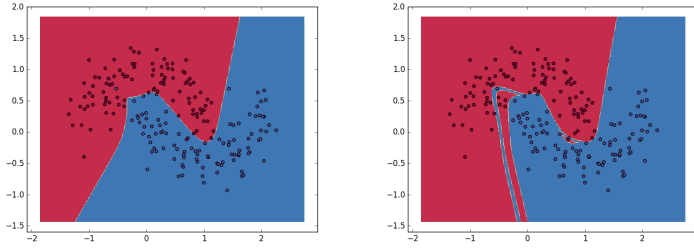


Figure 8: Exploring architecture with sigmoid (left: hidden layer [4, 10, 20, 30]; right: hidden layer [30, 20, 10, 4])

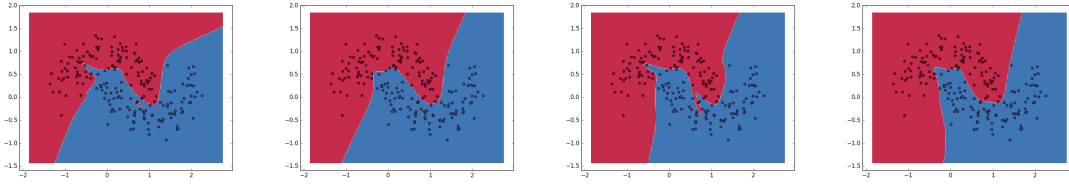


Figure 9: Exploring expressive power with tanh and sigmoid (left: tanh architecture 1; left middle: sigmoid architecture 1; right middle: tanh architecture 2; right: sigmoid architecture 2)

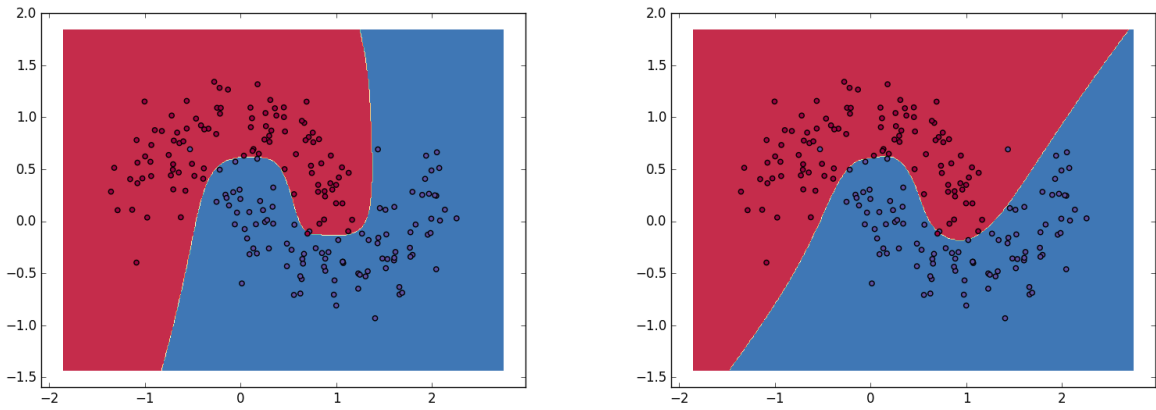


Figure 10: Exploring deep architecture with tanh activation (left: 9 layers of 6 nodes; right: 15 layers of 6 nodes)

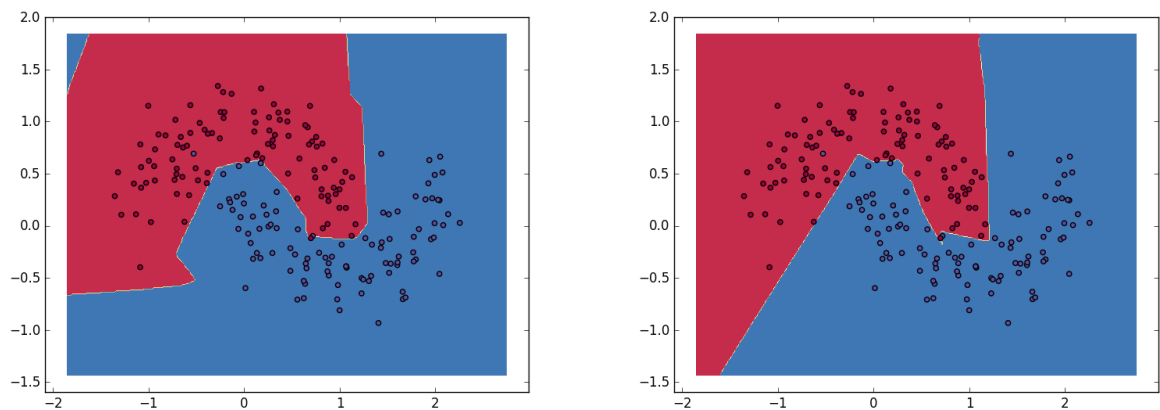


Figure 11: Exploring deep architecture with relu activation (left: 6 layers of 6 nodes; right: 9 layers of 6 nodes)

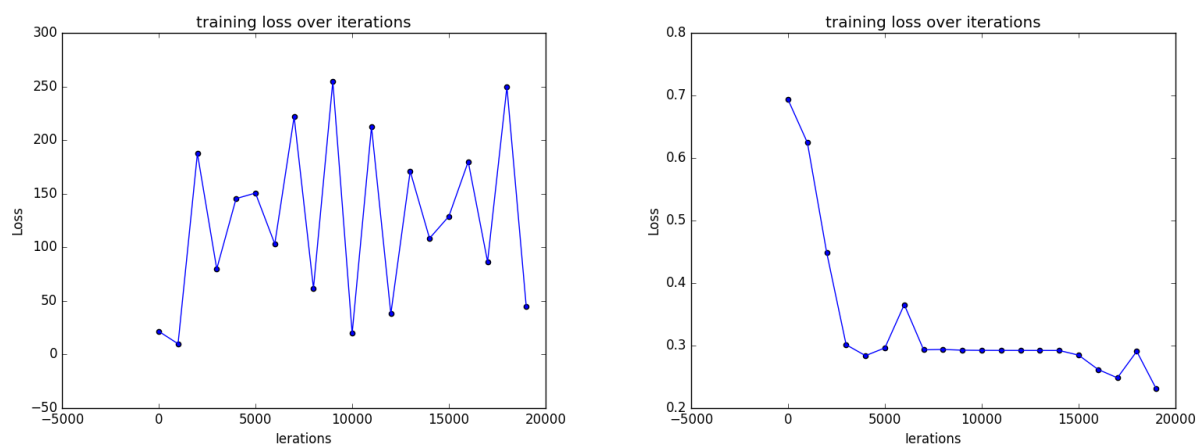


Figure 12: unnormalized model. left: model 1, right: model 2

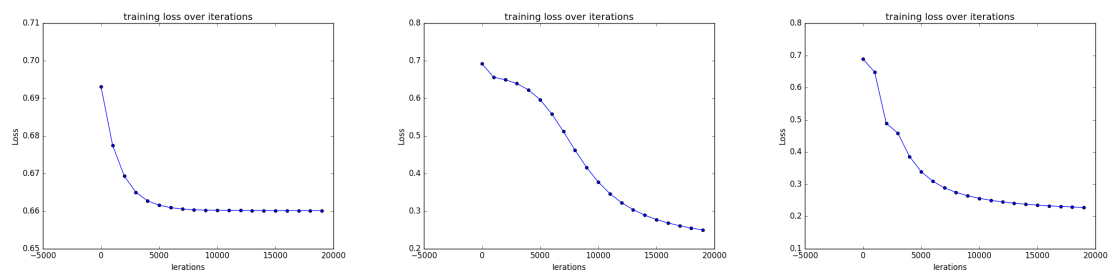


Figure 13: normalized model. left: model 3, middle: model 4, right: model 5

2 Training a Simple Deep Convolutional Network on MNIST

2.1 Build and Train a 4-layer DCN

2.1.1 Visualizing Training Loss

The training loss of the CNN is recorded here. As indicated, the loss quickly decays and entered the bottom of the curve and remained low for the rest of the iteration as seen in .

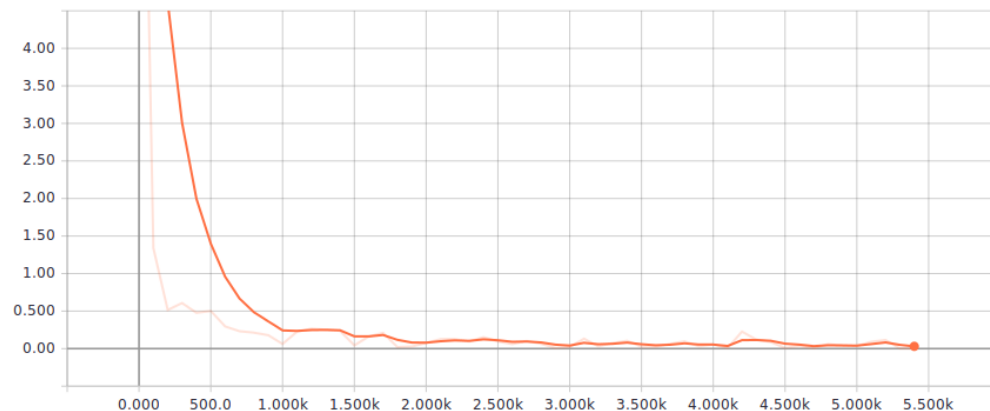


Figure 14: Cross Entropy of Model for Training data

2.2 More on Visualizing Training

In addition to prediction loss, I recorded max, min, mean, std for all matrices throughout calculation. Below are some of the scalar plots and histograms.

Ignore the noise background line that is not part of cross entropy, the model is doing very well outside of training set. It is almost achieving 100 percent accuracy. After few rounds of training, the accuracy quickly hovers to around 100 percent.

In figure 15, the mean of weights of convolutional layer one fluctuates and eventually goes down, as the range of weights increases as indicated in stddev, increasing max, and decreasing min. This is a sign of the weights learning different features.

In figure 16, due to the nature of ReLU, the activation cuts off at 0.

In figure 17, the bias of the matrices are very segmented. There are fixed number of nodes that are within certain range of value. Through training, each of these range of bias increase or decrease together.

In figure 18, the Weights Histogram looks health and wholesome. There is a well distribution of wide range of values and it follows somewhat a Gaussian distribution.

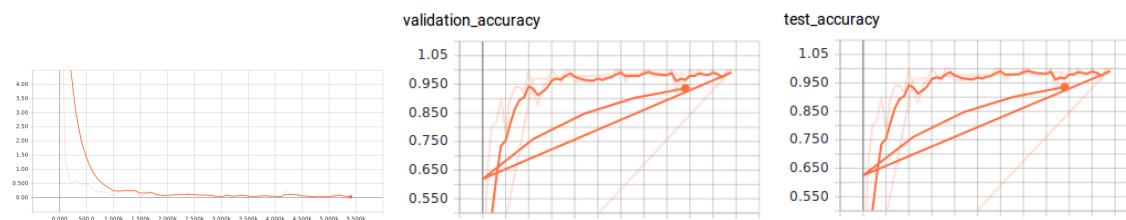


Figure 15: Cross Entropy of Model for Training (left), Validation (middle), and Testing data (right)

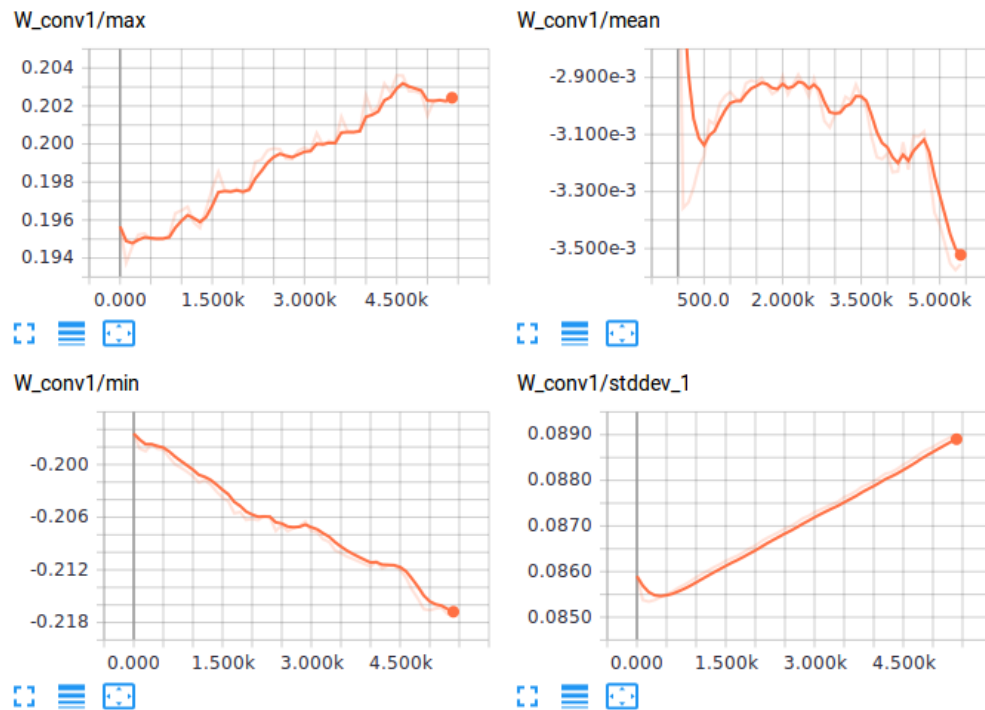


Figure 16: Statistics on Weights of first Convolutional Layer

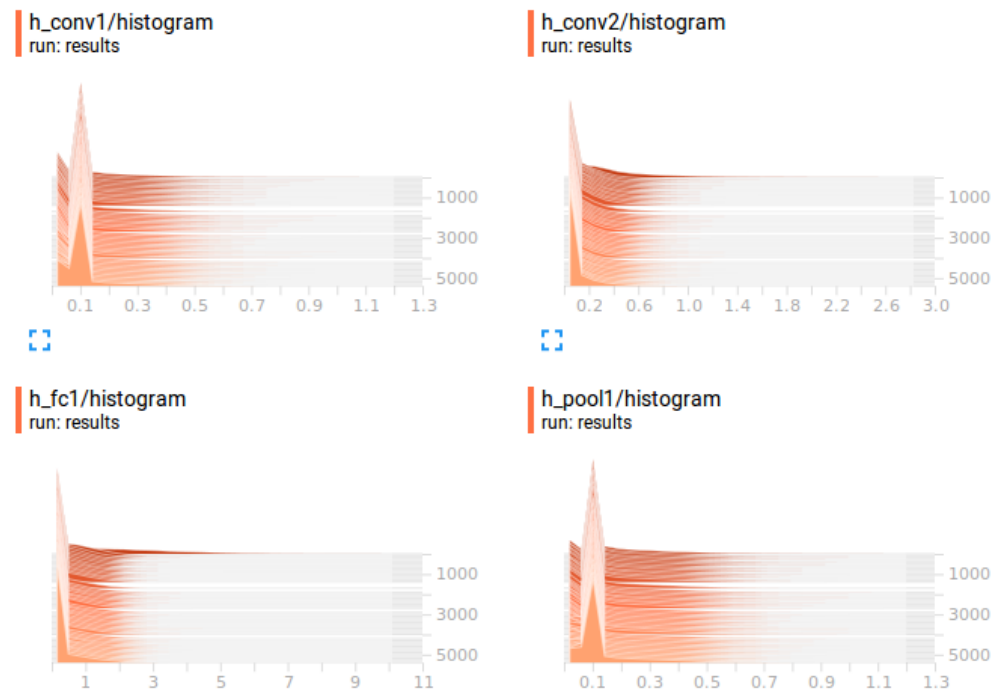


Figure 17: Histograms of different activations

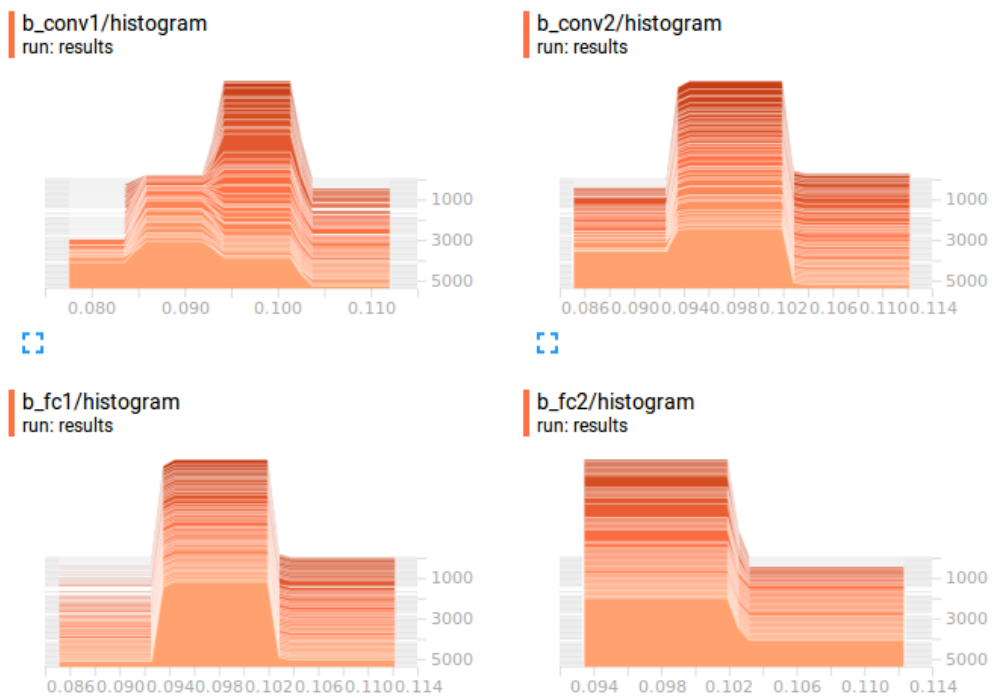


Figure 18: Histograms of bias matrices

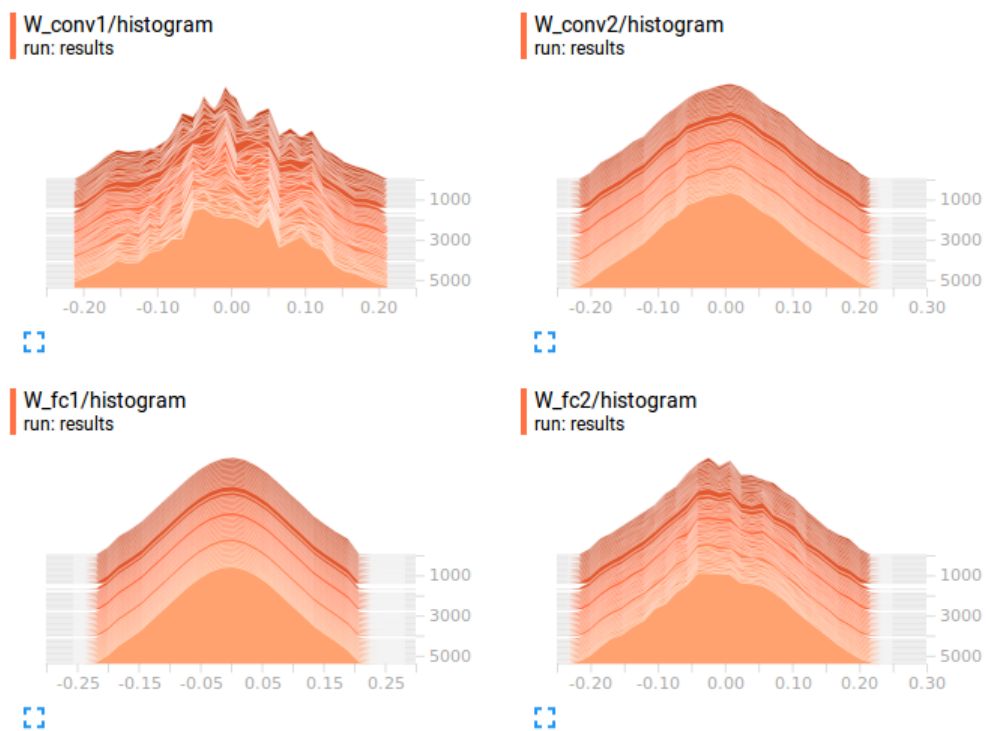


Figure 19: Histograms of Weights

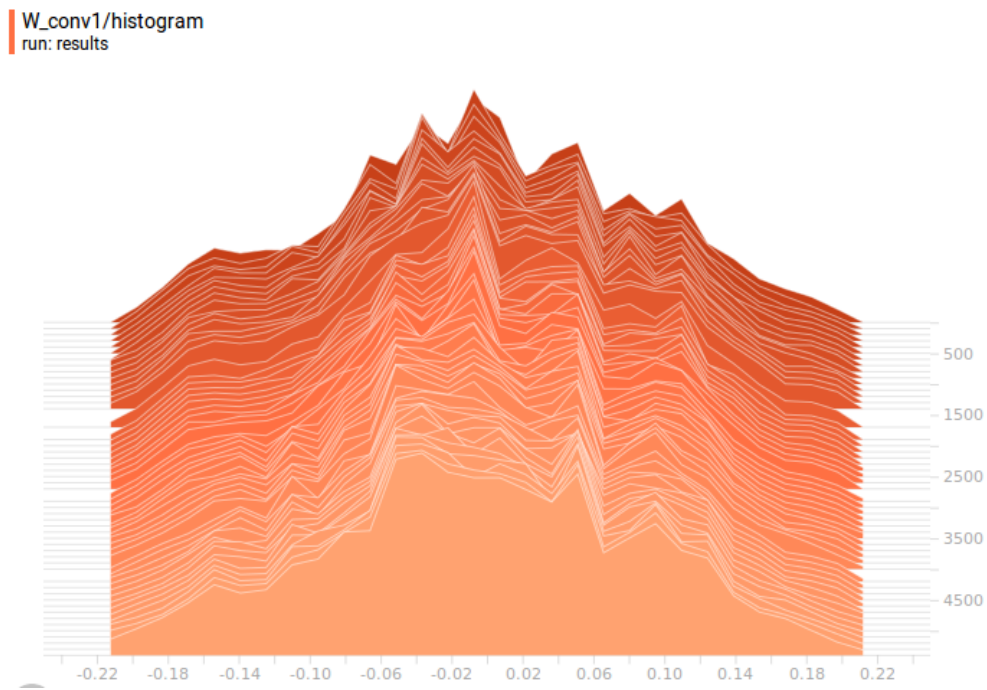


Figure 20: Histogram of Weights in Convolutional Layer 1

2.3 Time for More Fun!

Initially, more of the weights are around 0. As the model trains, the weights becomes more distributed.

In addition to normal CNN model, I used Tanh as activation instead of ReLU, and Momentum-based optimization strategy instead of Adam algorithm for optimization. The overall performance of the model is much worse than the original one, but a lot of it is due to potentially to small of momentum or learning rate it is set to that it has not converge to an optimal solution yet.

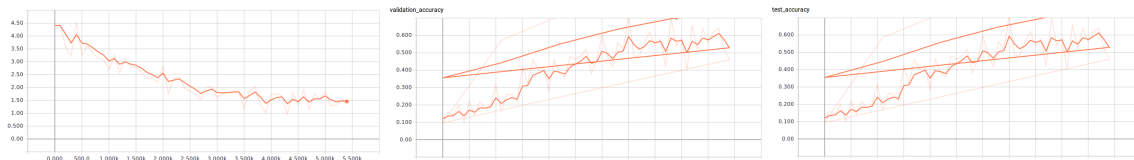


Figure 21: Cross Entropy of Model for Training (left), Validation (middle), and Testing data (right)

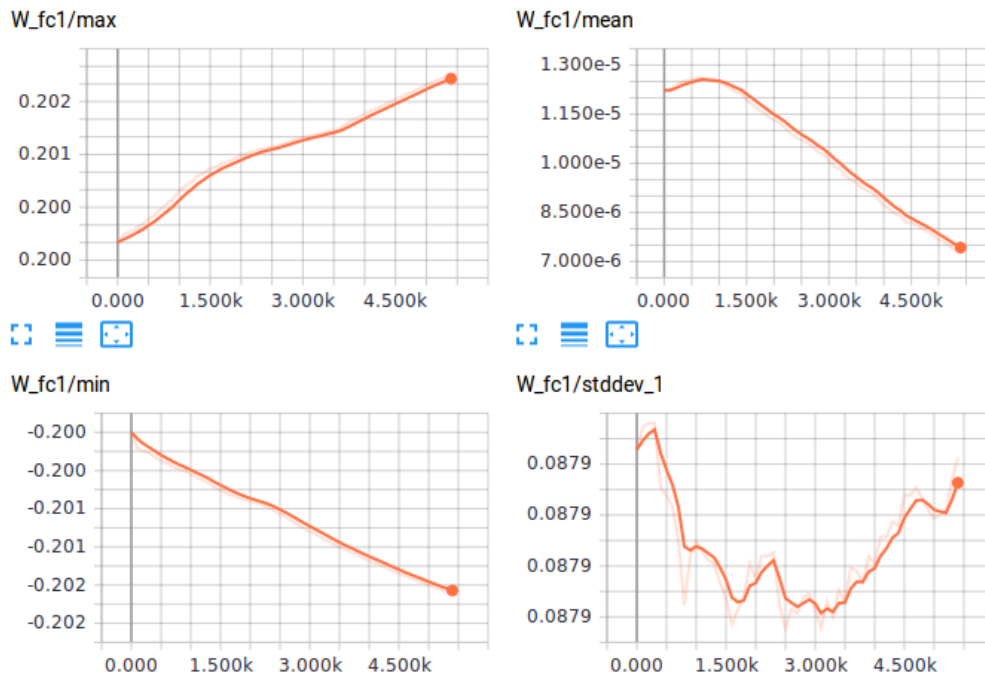


Figure 22: Weights of Fully Connected Layer 1

In figure 21, the mean of weights decreases gradually as model is being trained, making it look like Ankit Dr Dean Hutch.

In figure 22, the bias of convolutional layer 1 becomes more distributed over time towards smaller values.

In figure 23, the weights distribution of different weight matrix looks more like a triangle with hard angle and edges than the previous model

In figure 24, The activation of the fully connected layer one is mostly around -1 and 1, which make sense for tanh activation function

In figure 25, the input into activation function becomes more broadly distributed as it goes from earlier to later layers.

b_conv1/histogram
run: results

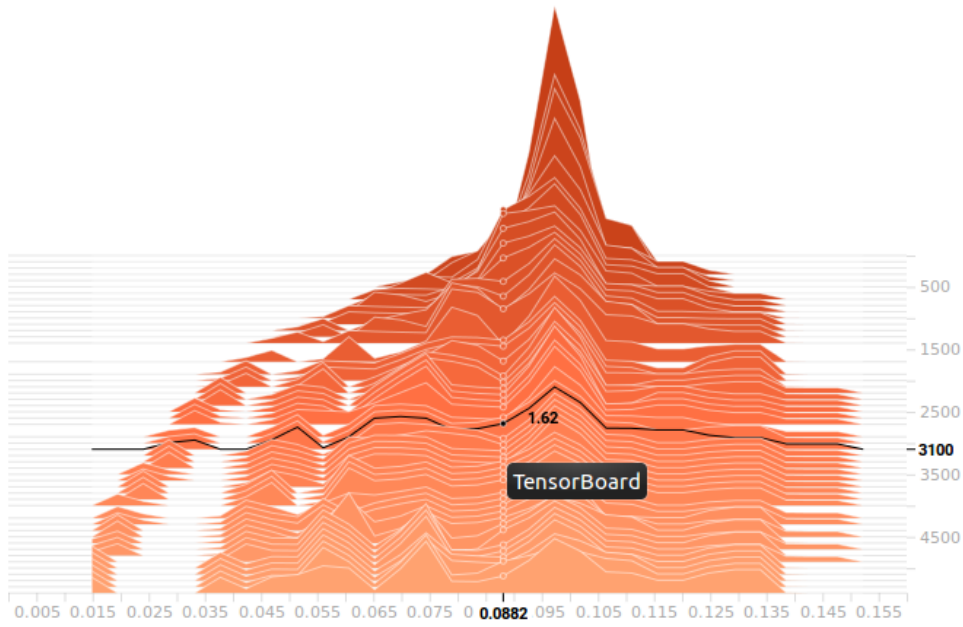
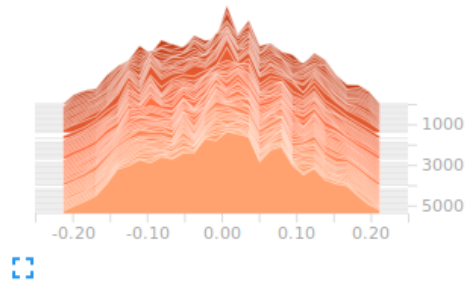
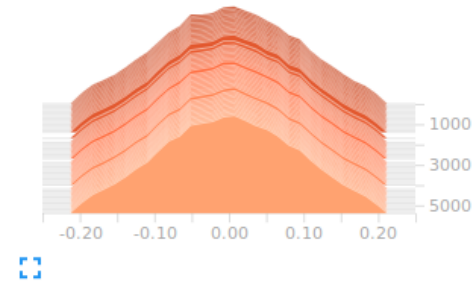


Figure 23: Histogram of bias in Convolutional Layer 1

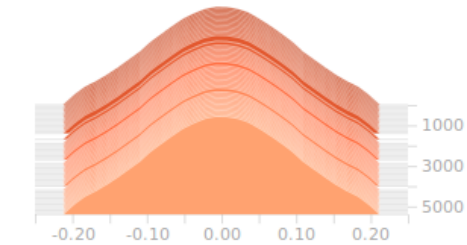
W_conv1/histogram
run: results



W_conv2/histogram
run: results



W_fc1/histogram
run: results



W_fc2/histogram
run: results

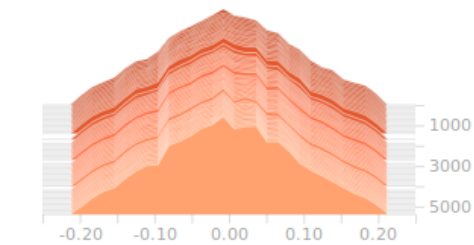


Figure 24: Histogram of Weights in Convolutional Layer 1

h_fc1/histogram
run: results

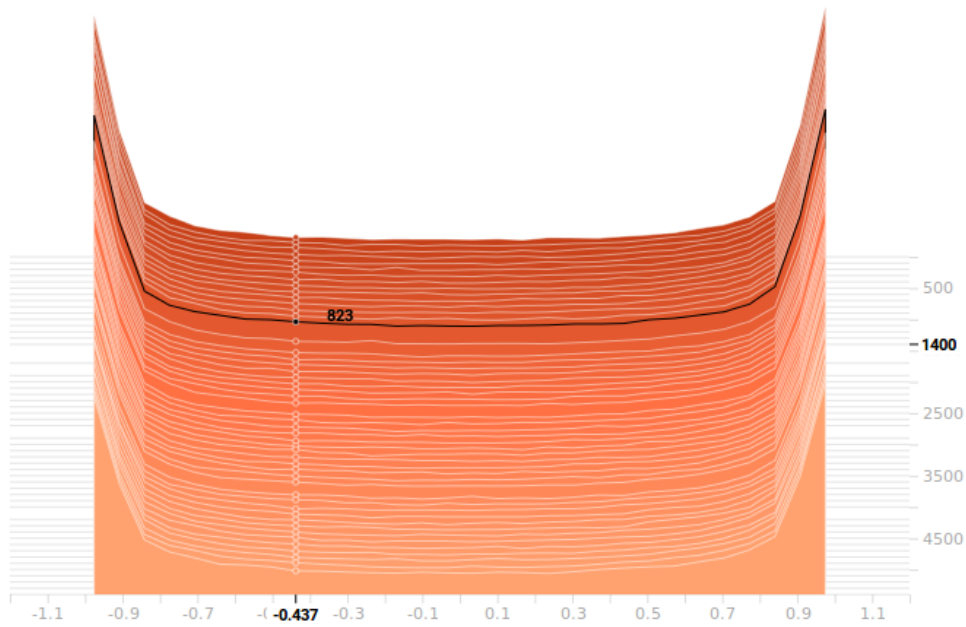
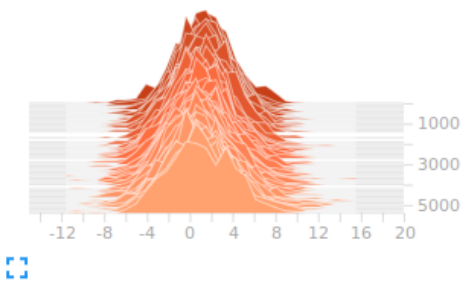
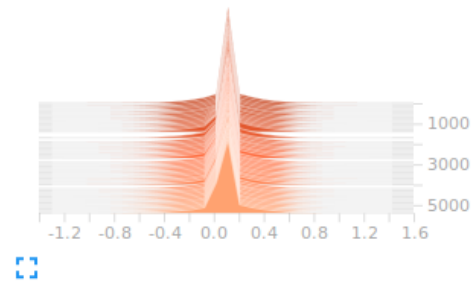


Figure 25: Histogram of activations in fully connected Layer 1

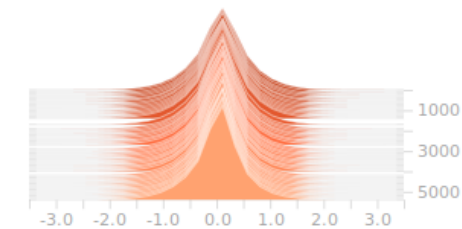
y_conv/histogram
run: results



z_conv1/histogram
run: results



z_conv2/histogram
run: results



z_fc1/histogram
run: results

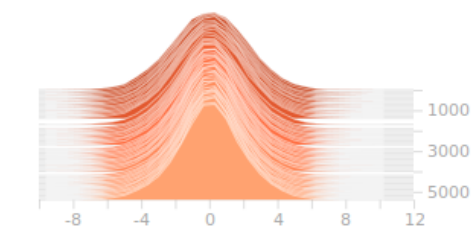


Figure 26: Histogram of input into activation functions

2.4 Comparison between models

The first model performed better than the second because it uses ReLU as its activation function, and for some reason, ReLU is able to make CNN better. In addition, Adam optimizer is recommended as a standard. Momentum optimizer may not be a good fit for this particular problem because the landscape is not optimal for the algorithm. Maybe there are a lot of changes in direction for the gradient update, causing little momentum to build up, lead to slow convergence to optimal solution.