COMP 576 Assignment 1

Luis Clague & Grace Tan Fall 2019

Contents

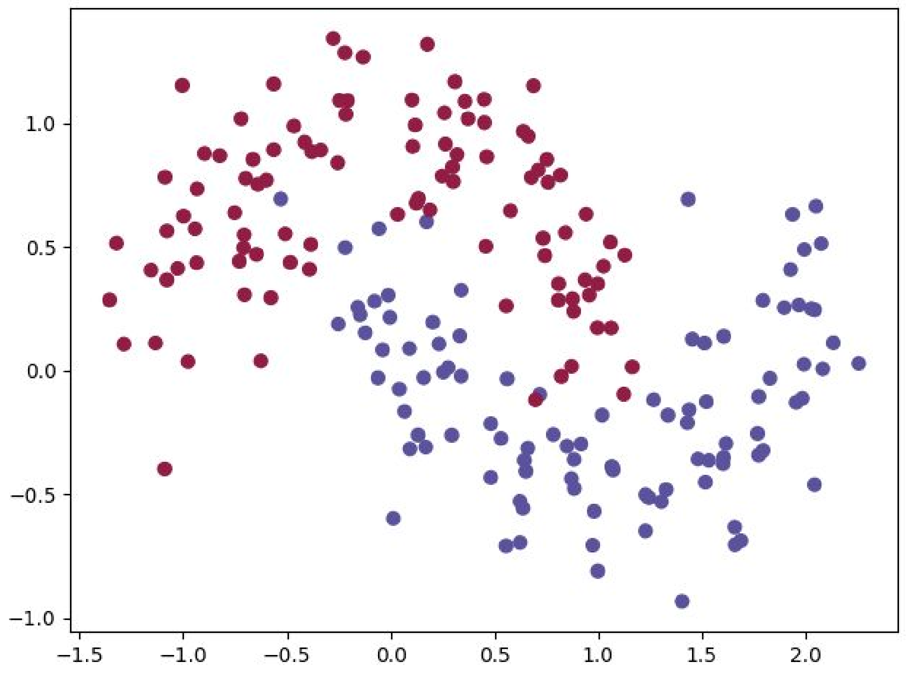
NOTE

This report was written by both Grace Tan (gzt1) and Luis Clague (lc61). We did this because of the piazza post: https://piazza.com/class/jzsvoj2yhjc4id?cid=30 which gave us express permission to do so. We also decided to submit the same report twice, due to the ambiguity in grading. This work represents both of our contributions.

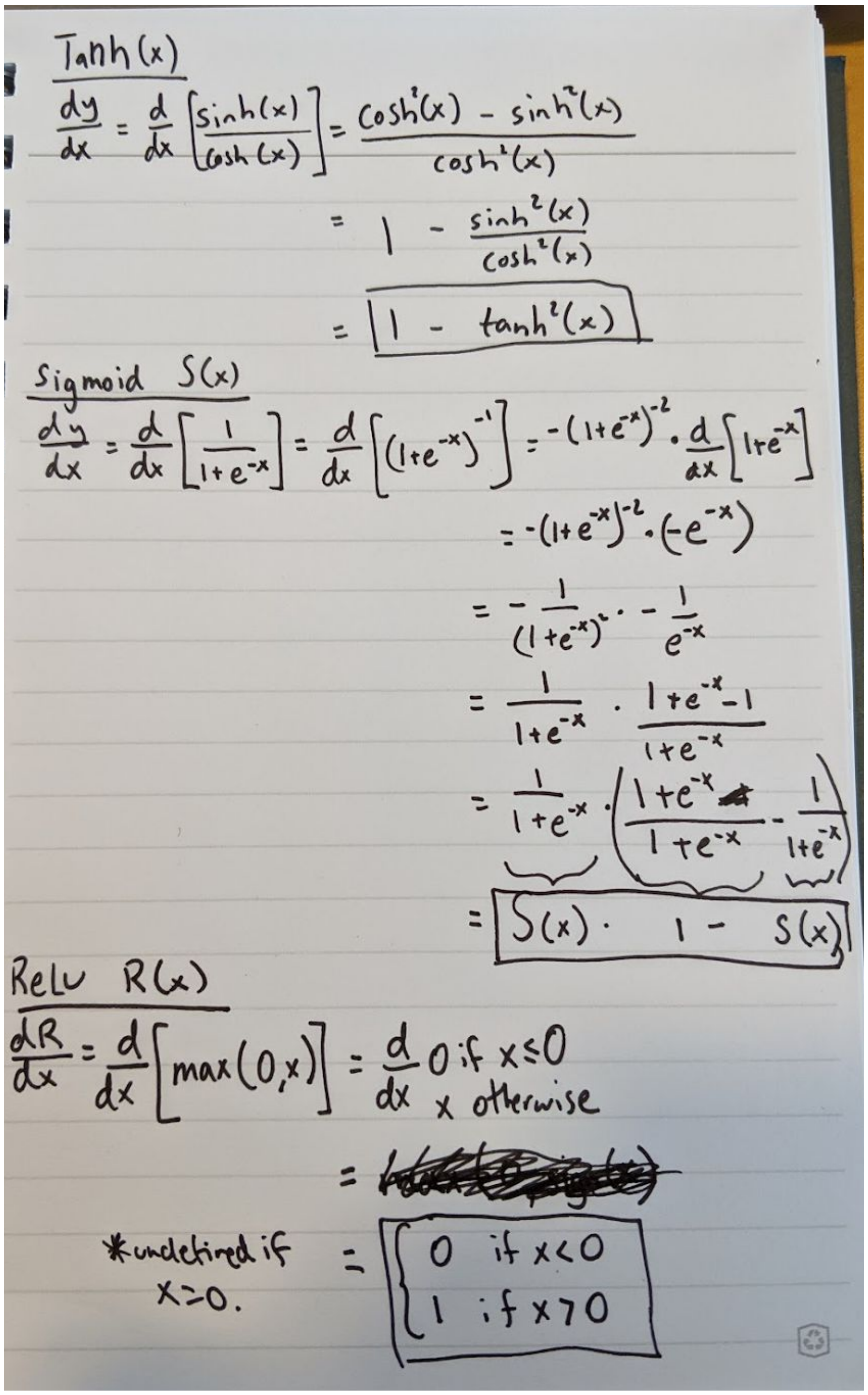
The respective code for this assignment can be found at https://github.com/clague17/comp576/hw01

1. Backprop in a Simple (3 layer) Neural Net

a. Dataset



b. Activation Functions



Tanh(x)

$$\frac{dy}{dx} = \frac{d}{dx}\left[\frac{\sinh(x)}{\cosh(x)}\right] = \frac{\cosh^2(x) - \sinh^2(x)}{\cosh^2(x)}$$

$$= 1 - \frac{\sinh^2(x)}{\cosh^2(x)}$$

$$= \boxed{1 - \tanh^2(x)}$$

Sigmoid S(x)

$$\frac{dy}{dx} = \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] = \frac{d}{dx}\left[(1+e^{-x})^{-1}\right] = -(1+e^{-x})^{-2} \cdot \frac{d}{dx}\left[1+e^{-x}\right]$$

$$= -(1+e^{-x})^{-2} \cdot (-e^{-x})$$

$$= -\frac{1}{(1+e^{-x})^2} \cdot -\frac{1}{e^{-x}}$$

$$= \frac{1}{1+e^{-x}} \cdot \frac{1+e^{-x}-1}{1+e^{-x}}$$

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

$$= \boxed{S(x) \cdot \quad 1 - \quad S(x)}$$

ReLU R(x)

$$\frac{dR}{dx} = \frac{d}{dx}\left[\max(0,x)\right] = \frac{d}{dx}\begin{matrix} 0 \text{ if } x \le 0 \\ x \text{ otherwise} \end{matrix}$$

= [illegible]

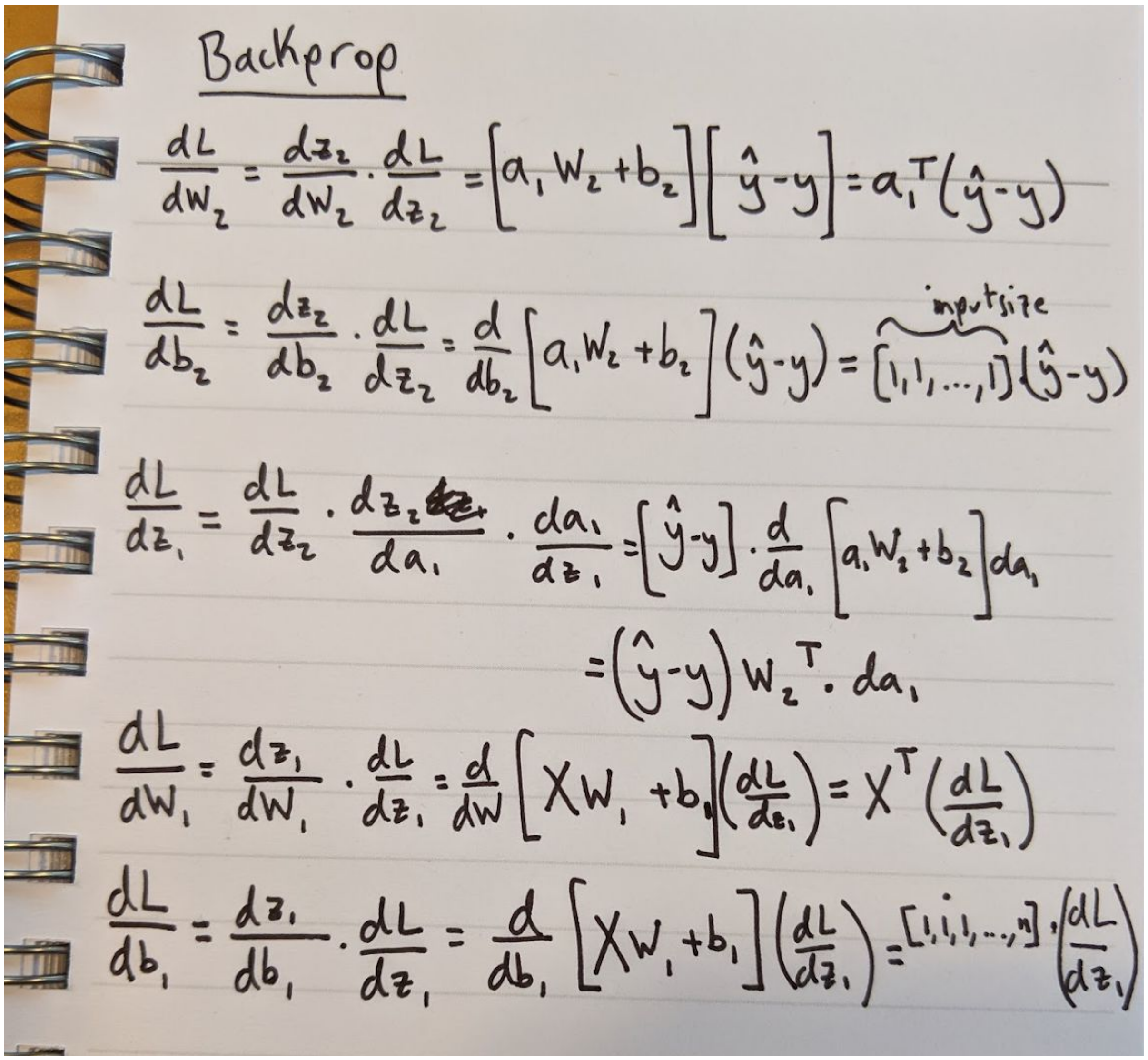
*undefined if $x=0$.

$$= \boxed{\begin{cases} 0 & \text{if } x<0 \\ 1 & \text{if } x>0 \end{cases}}$$

c. Building the Net

See the code :)

d. Implementing Backprop



Backprop

$$\frac{dL}{dW_2} = \frac{dz_2}{dW_2} \cdot \frac{dL}{dz_2} = \left[a_1 W_2 + b_2\right]\left[\hat{y} - y\right] = a_1^T(\hat{y} - y)$$

$$\frac{dL}{db_2} = \frac{dz_2}{db_2} \cdot \frac{dL}{dz_2} = \frac{d}{db_2}\left[a_1 W_2 + b_2\right](\hat{y} - y) = \overbrace{[1,1,\ldots,1]}^{\text{inputsize}}(\hat{y} - y)$$

$$\frac{dL}{dz_1} = \frac{dL}{dz_2} \cdot \frac{dz_2}{da_1} \cdot \frac{da_1}{dz_1} = \left[\hat{y} - y\right] \cdot \frac{d}{da_1}\left[a_1 W_2 + b_2\right] da_1$$

$$= (\hat{y} - y) W_2^T \cdot da_1$$

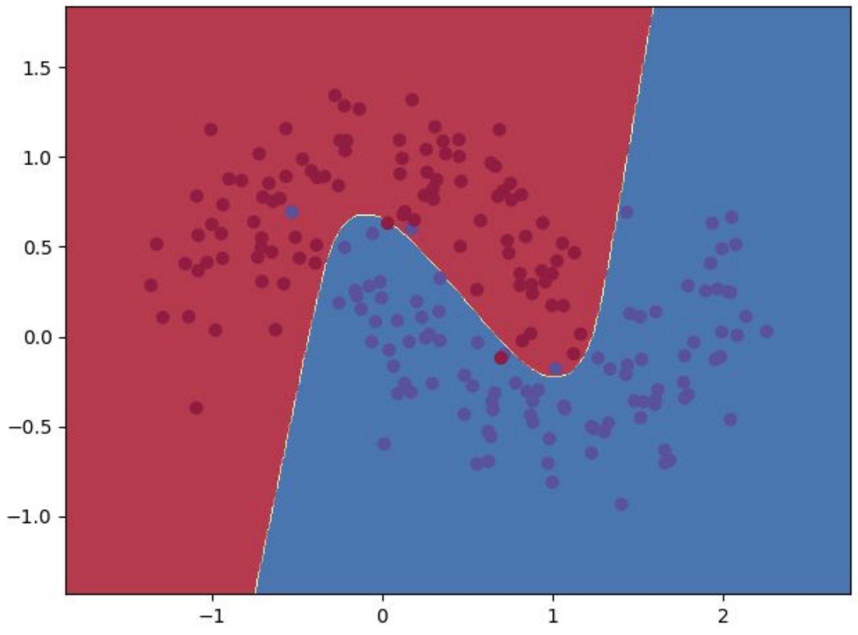
$$\frac{dL}{dW_1} = \frac{dz_1}{dW_1} \cdot \frac{dL}{dz_1} = \frac{d}{dW}\left[X W_1 + b_1\right]\left(\frac{dL}{dz_1}\right) = X^T\left(\frac{dL}{dz_1}\right)$$

$$\frac{dL}{db_1} = \frac{dz_1}{db_1} \cdot \frac{dL}{dz_1} = \frac{d}{db_1}\left[X W_1 + b_1\right]\left(\frac{dL}{dz_1}\right) = [1,1,1,\ldots,n] \cdot \left(\frac{dL}{dz_1}\right)$$

e. Training

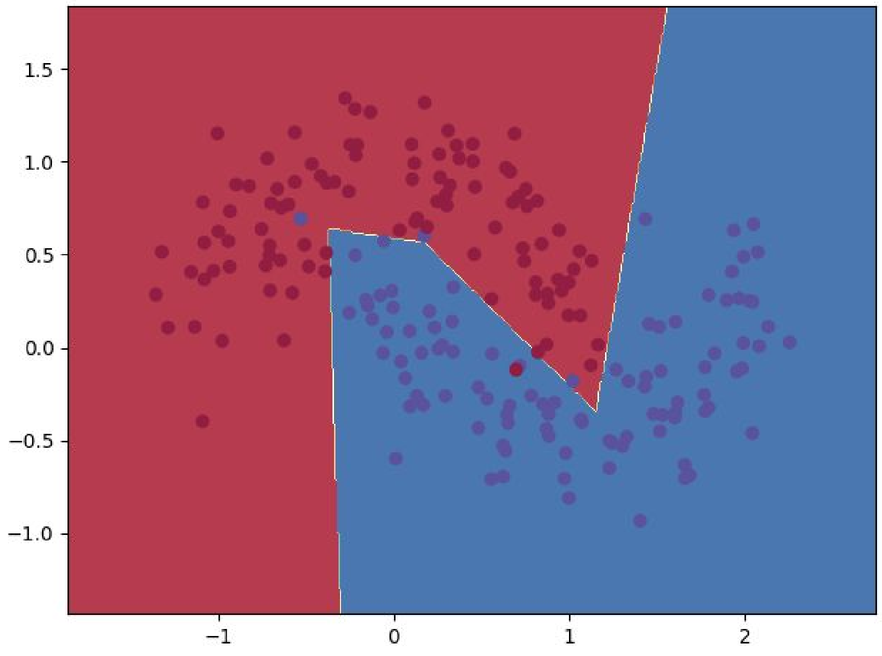
i. tanh

Loss after iteration 19000: 0.070758



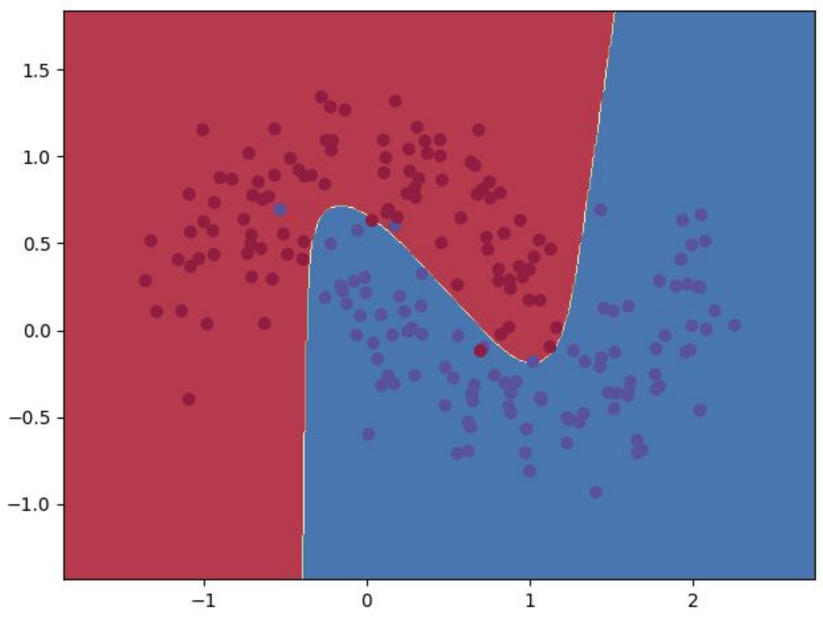
ii. Relu

Loss after iteration 19000: 0.071219



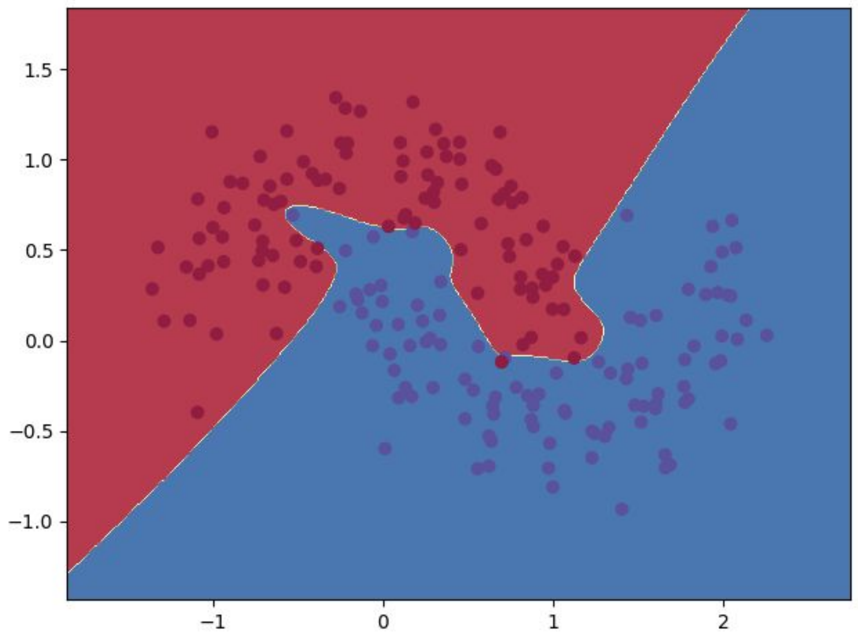
iii. Sigmoid

Loss after iteration 19000: 0.078794



iv. tanh 150 neurons

Loss after iteration 19000: 0.035834



General Observations:
The decision boundary for all three are relatively similar, complete with relatively similar error loss after 1900 iterations. The notable exception is the relu which is obviously a lot sharper than the others since it is a simple piecewise function. When I tuned the number of neurons to 150 for the tanh activation function, the decision boundary drastically overfit compared to only 3 neurons. This is also reflected by a “better” result, in that the loss was much smaller at .035834

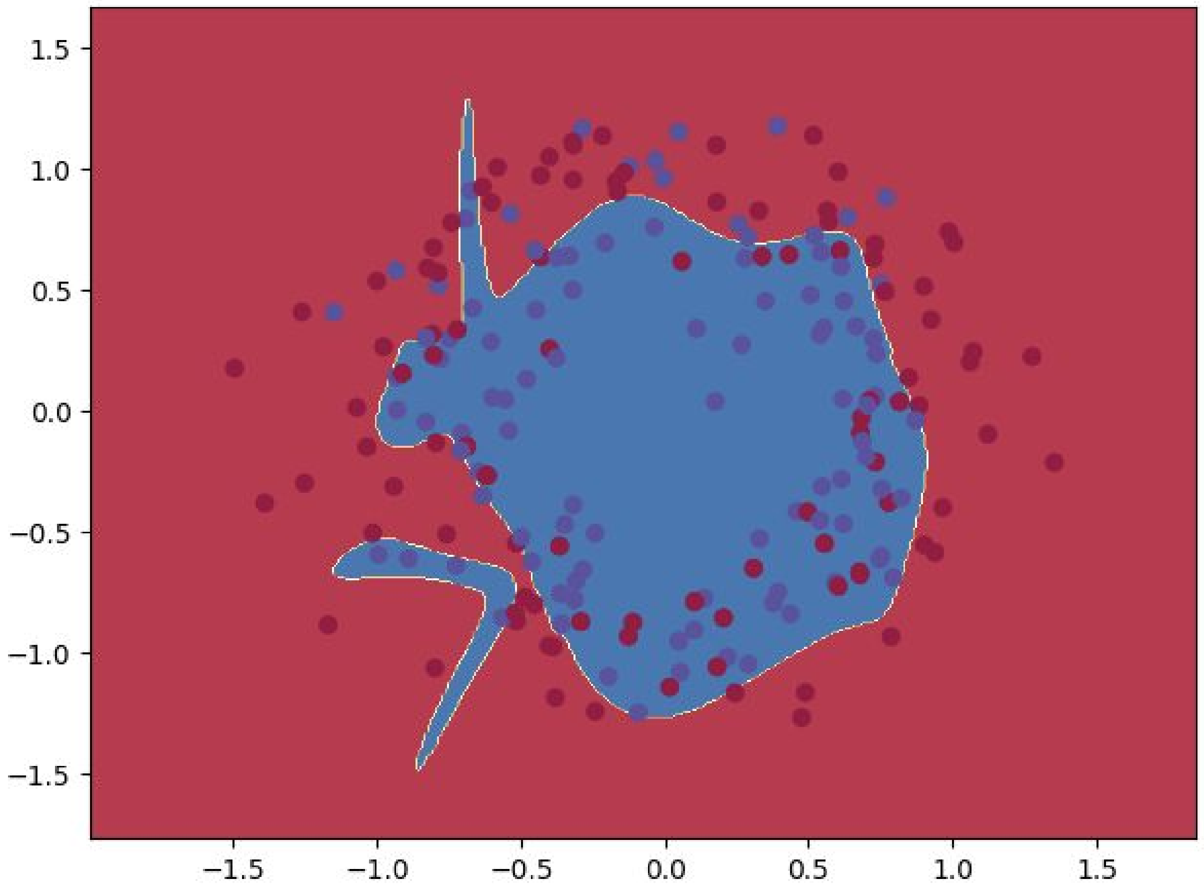
2. Training a Deeper Net

a. Design

FinalLayer vs (Hidden) Layer
I decided to separate the neural net into two distinct classes. One for a regular layer “Layer” and one for a “FinalLayer”, in order to separate the variant and invariant parts of this system. For a FinalLayer object, the backprop step is a bit different being that we have to reduce the weight vector into one approximation, and not pass on the delta like we do in a regular hidden Layer.
I also decided to create my system this way such that during the backprop logic in the greater NeuralNet class, we can just delegate to each composee and not muddle the already complex class with even more complexity.
My neural network class can also specify the shape of the layers, and of course the activation function for it.

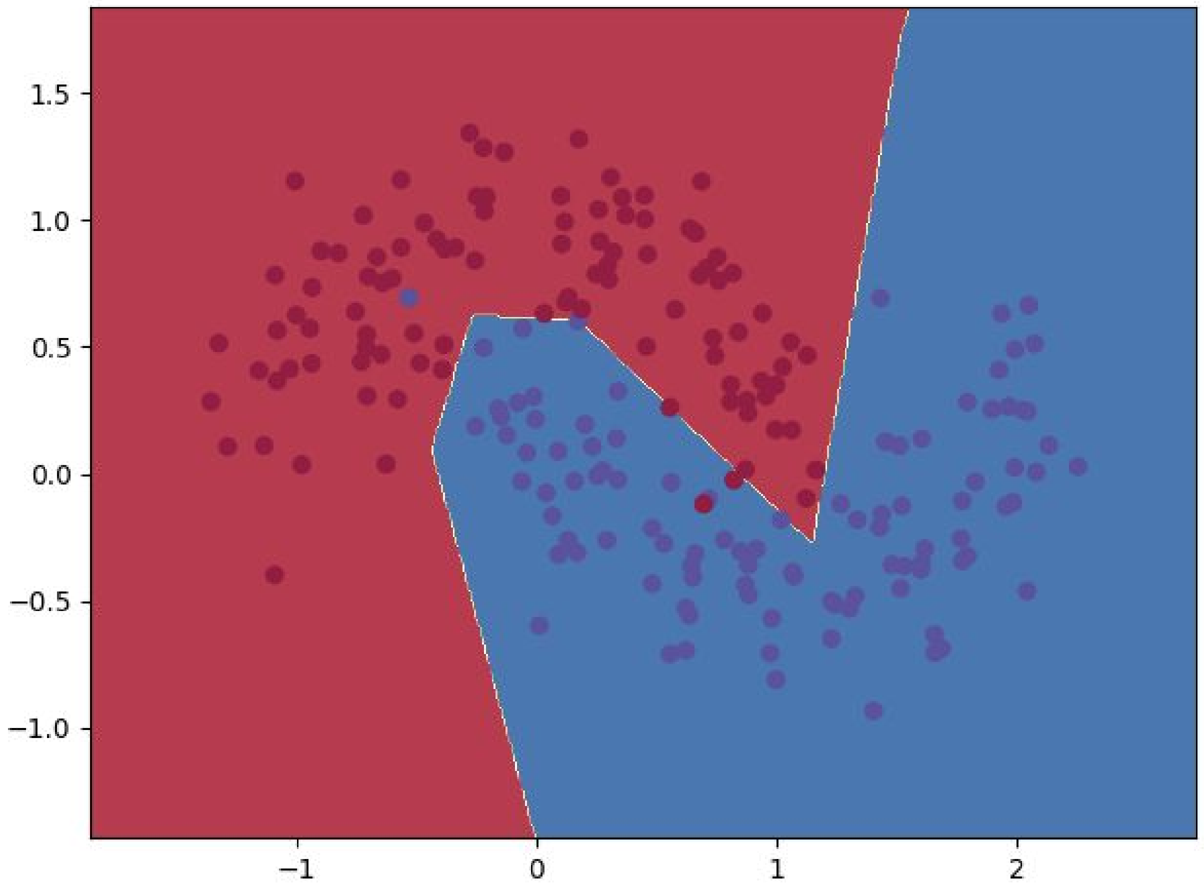
Below are some examples with different hidden layers and different activation functions.

b. tanh 5 / 5 / 5 Make Circles



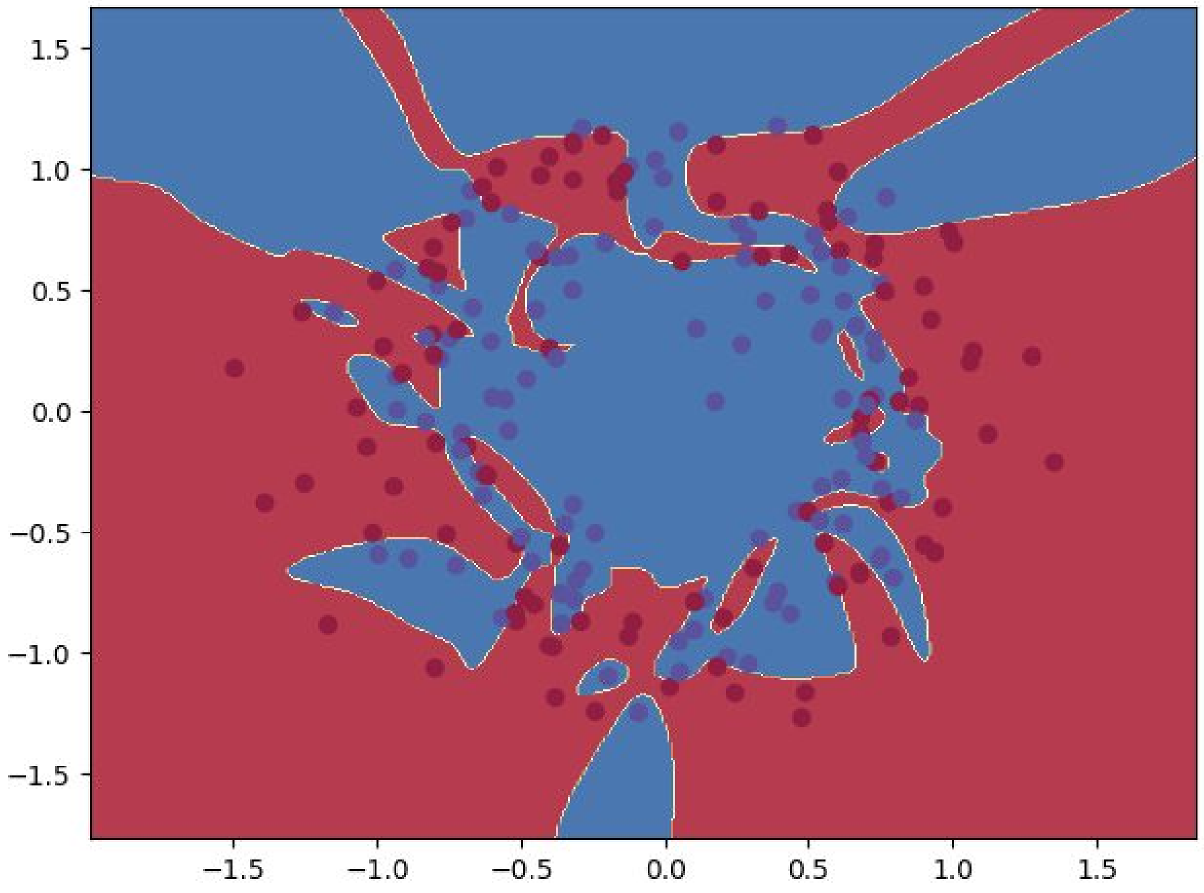
With 3 hidden layers, the model is severely overfitting, and not really able to accurately predict the dataset. The loss after 49000 iterations is shockingly high at 0.437079, and this is definitely not the best for the data set.

c. Relu 4 / 4 / 4 / 4 / 5



It seems like adding a couple extra layers for the neural net in this case helped, as it was able to reduce the loss bit a bit when compared to our original 3 layer net. After 49000 iterations, the loss was 0.069296, lower than the .071219 across 1900 iterations in our previous experiment.

d. tanh 30 / 30 Make Circles

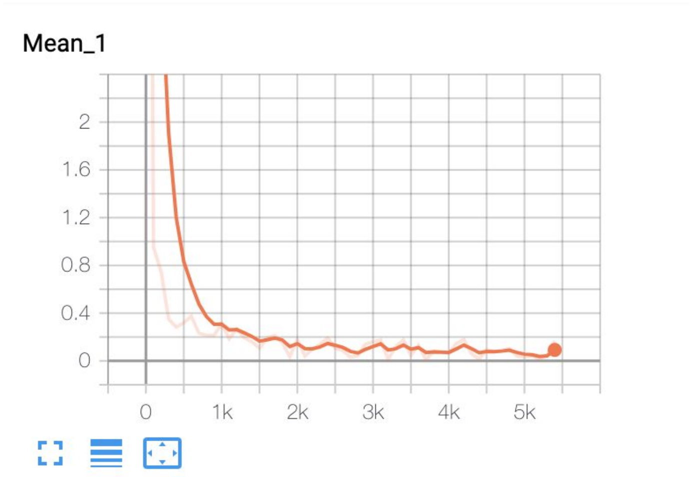


This was a much harder dataset to fine tune. The tanh function seemed to respond best to more neurons and less layers, resulting in the extremely good, yet extremely overfitted function you see here. This had a loss of .034260 after 49000 iterations. The decision boundary was incredibly jagged, in order to include/exclude the most points which is what resulted in such a small loss function.

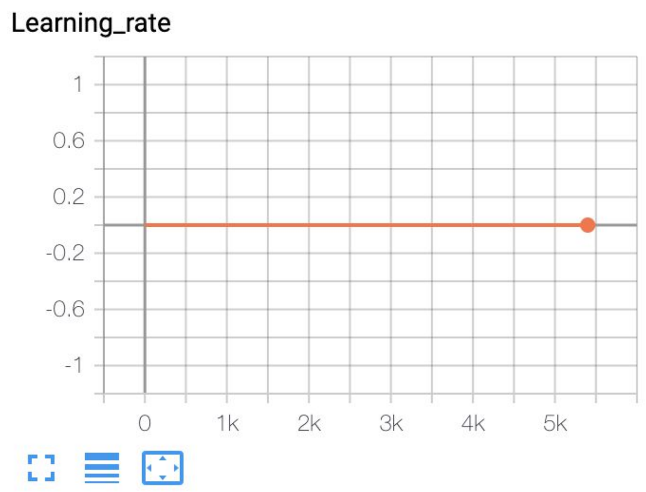
3. Training a simple CNN on MNIST

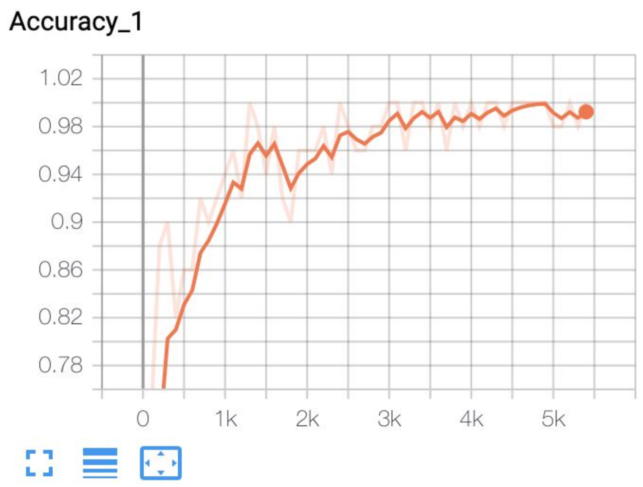
a. Building a 4-layer DCN

My test accuracy: .9858 which took 391.007981 seconds to finish
*** Note, the label for the following graphic should be Loss, not Mean_1.

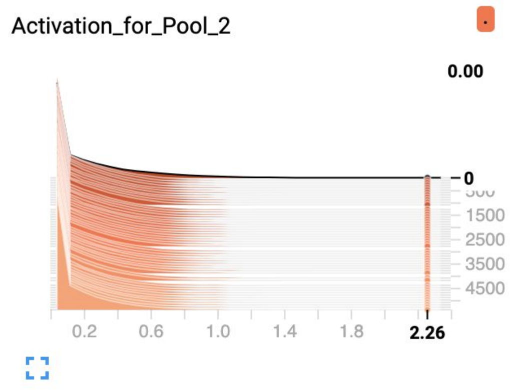


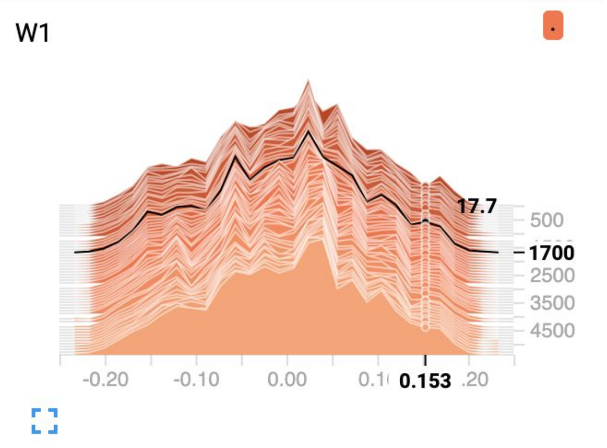
i. Running Training

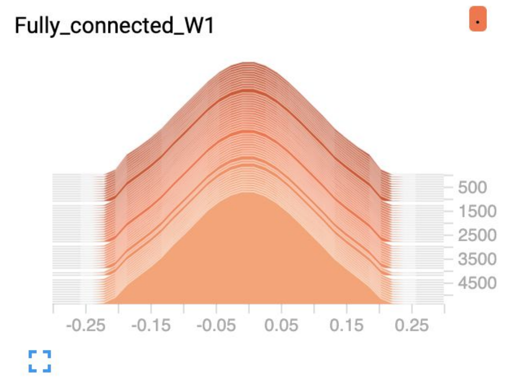


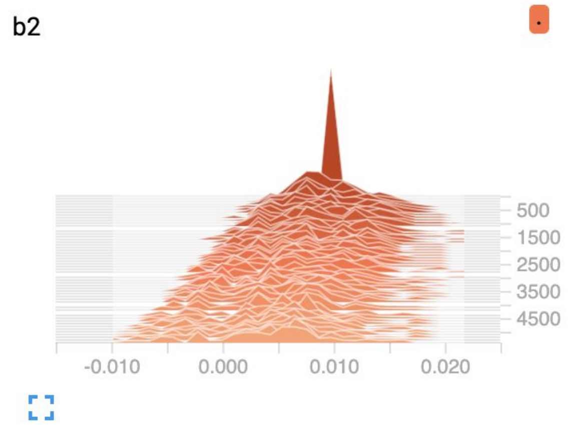


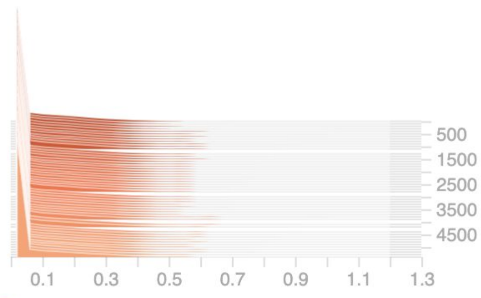
ii. More Visualization of Training

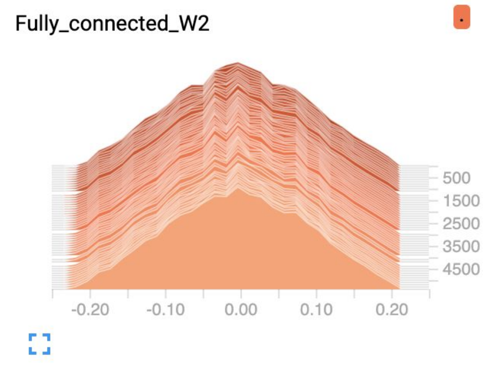


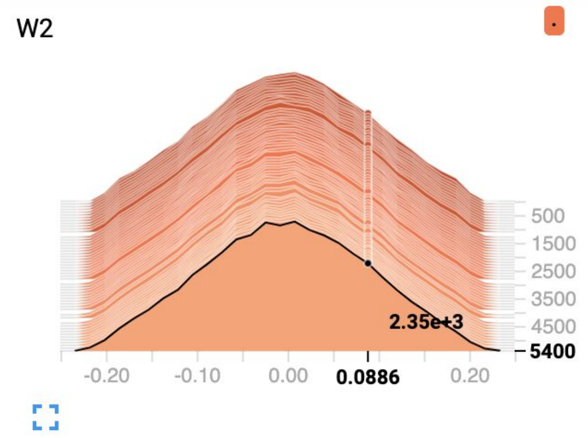


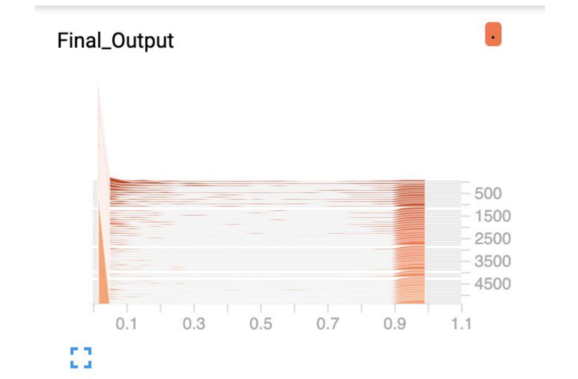


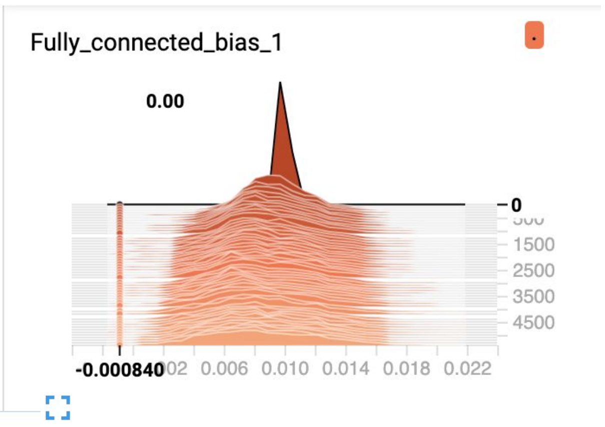


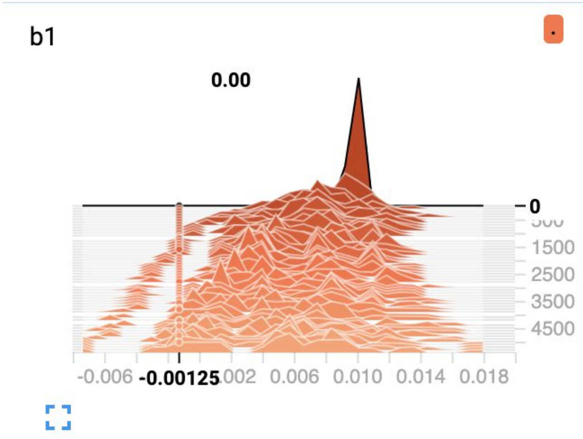








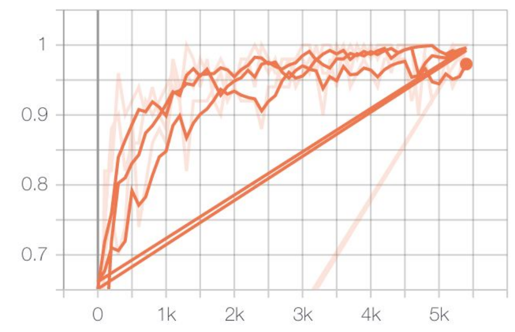




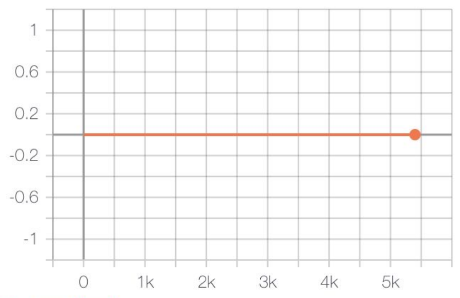
iii. Time for more fun

Changed Activations to abs value:

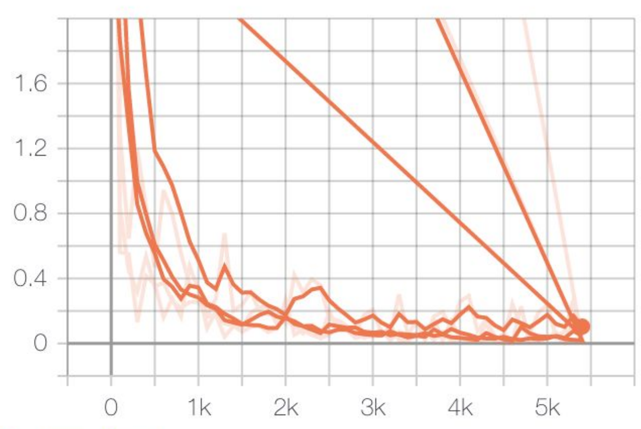
Accuracy_1



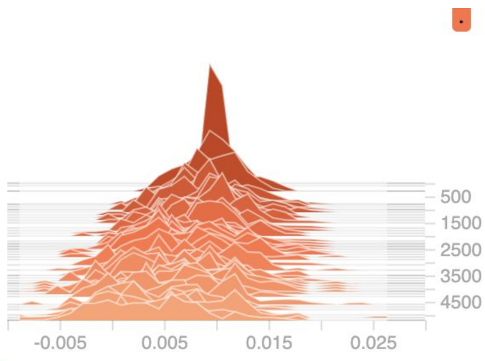
Learning_rate



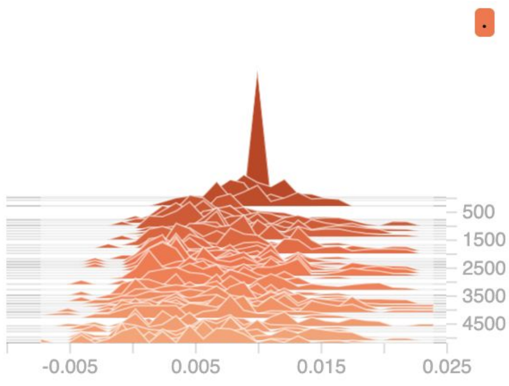
Loss



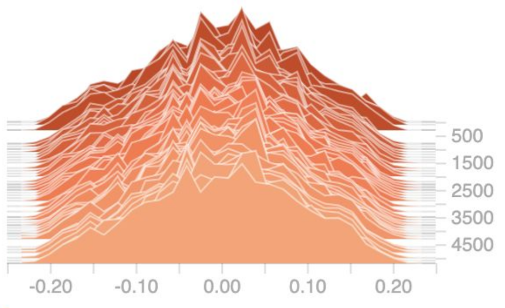
b2



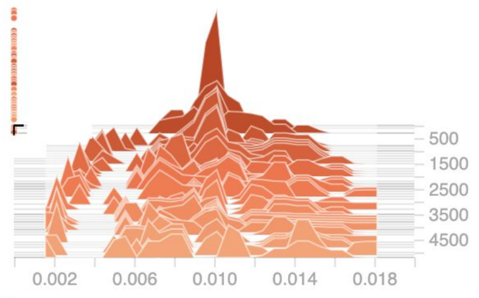
b1



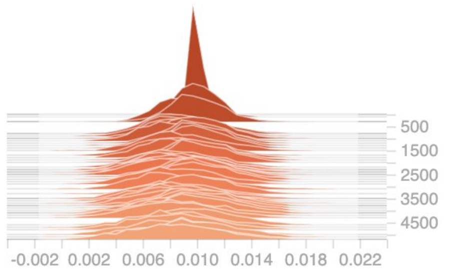
W1



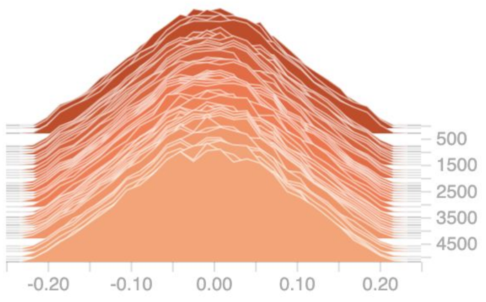
Fully_connected_bias_1_1



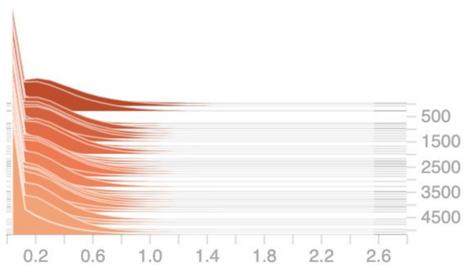
Fully_connected_bias_1



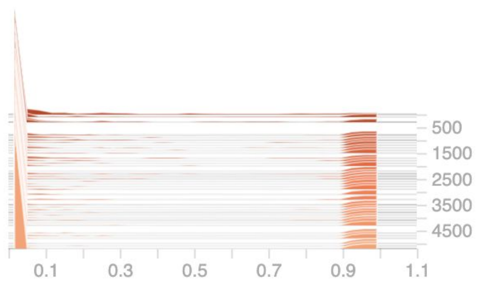
Fully_connected_W2

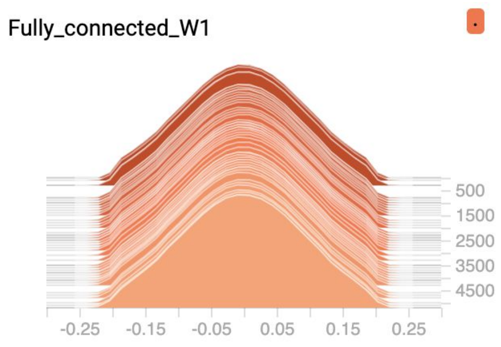


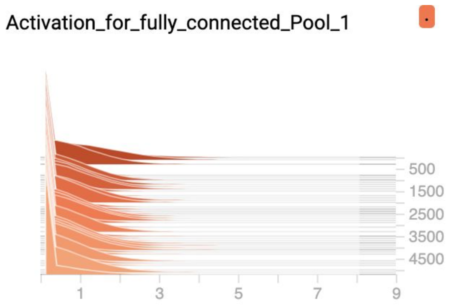
Activation_for_Pool_2

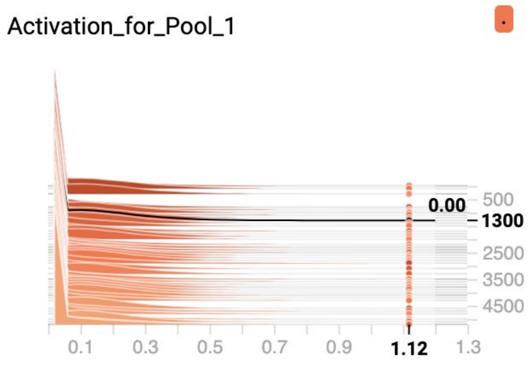


Final_Output



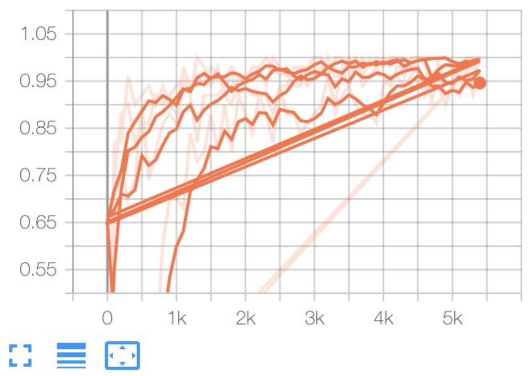




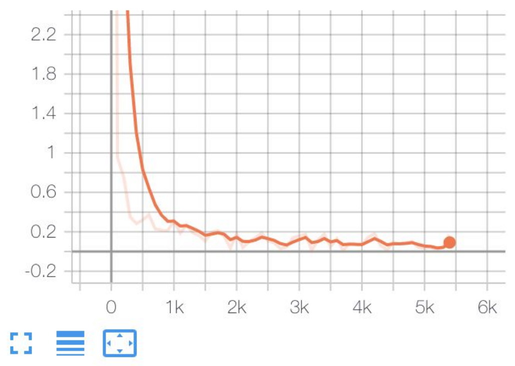


Changed Activations to sigmoid:

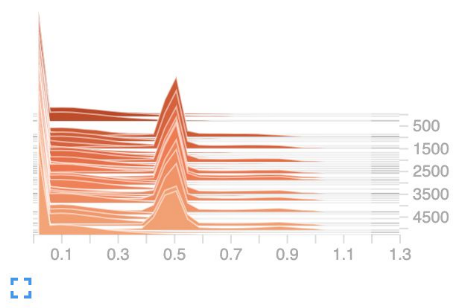
Accuracy_1



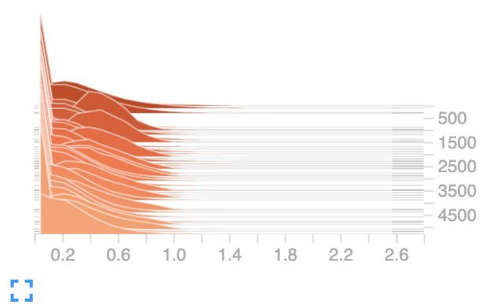
Mean_1



Activation_for_Pool_1

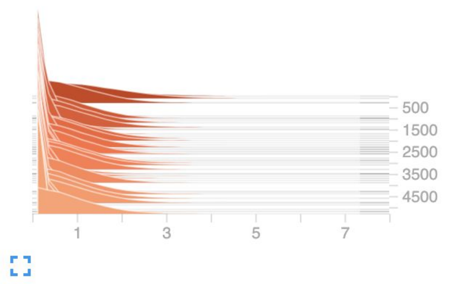


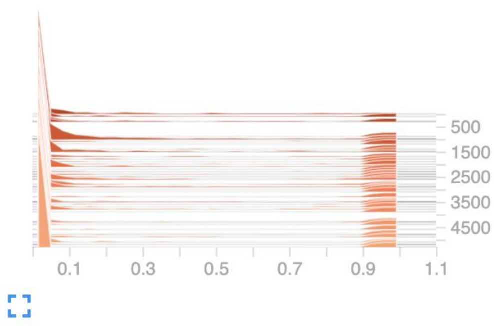
Activation_for_Pool_2



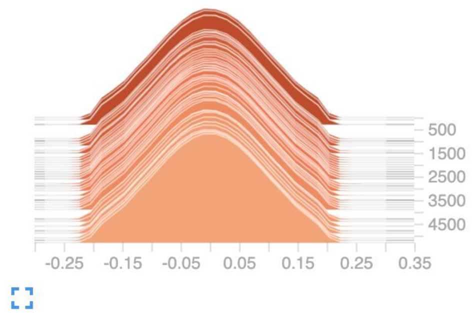
Final_Output

Activation_for_fully_connected_Pool_1

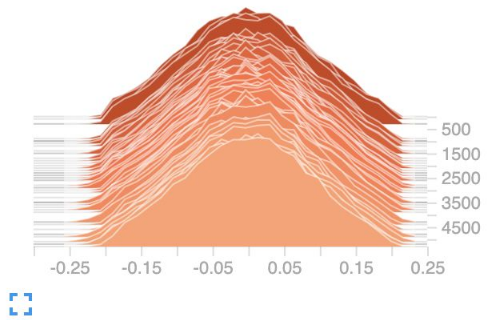


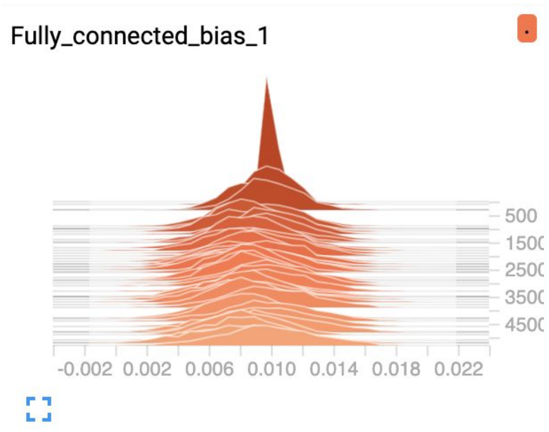


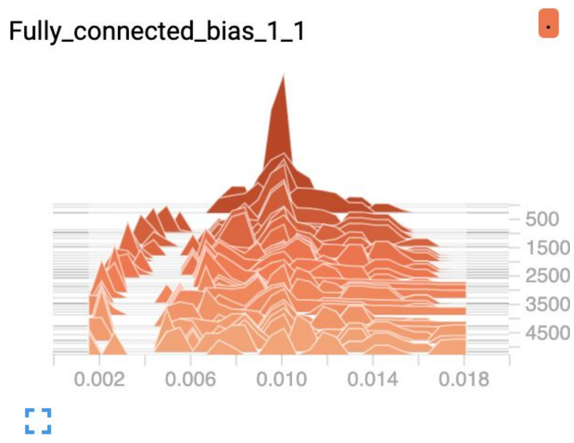
Fully_connected_W1

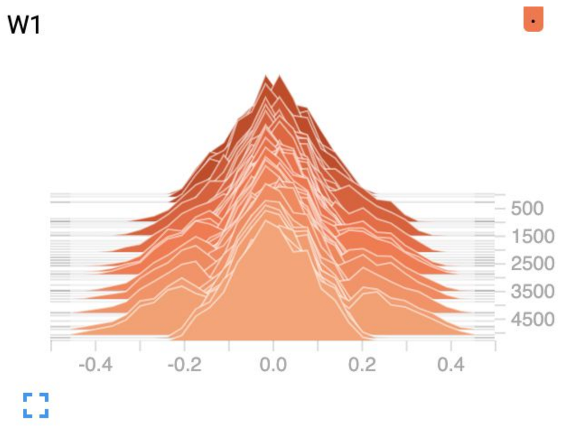


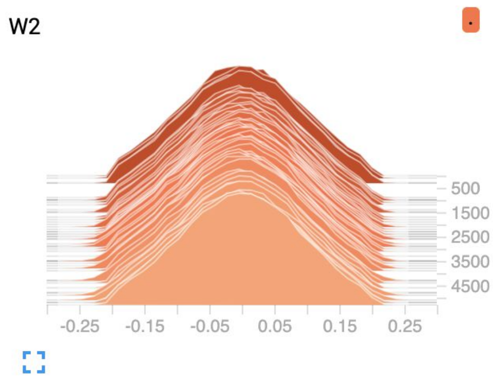
Fully_connected_W2

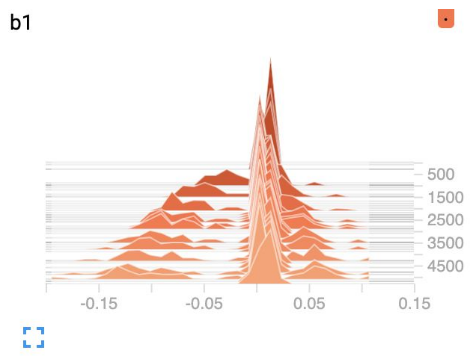


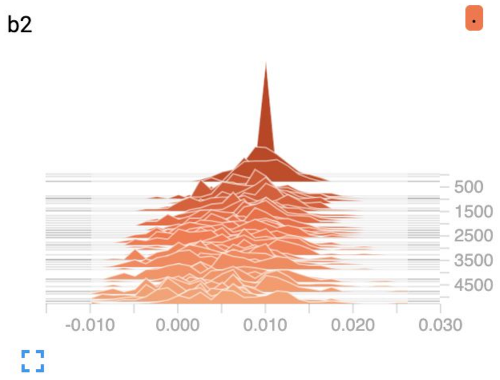












4. Sources

Lots of credit goes to https://gist.github.com/saitodev/c4c7a8c83f5aa4a00e93084dd3f848c5 for helping me to debug certain tf things, as well as providing me with a sanity check while training my models.