Homework 1

Joshua Michalenko
ELEC 677: Deep Learning
Dr. Ankit Patel

10/04/16

1 Problem 1: Backpropogation in a simple Nueral Network

1.1 Part A: Dataset

The plot of the Two Moons dataset is displayed in Figure 1

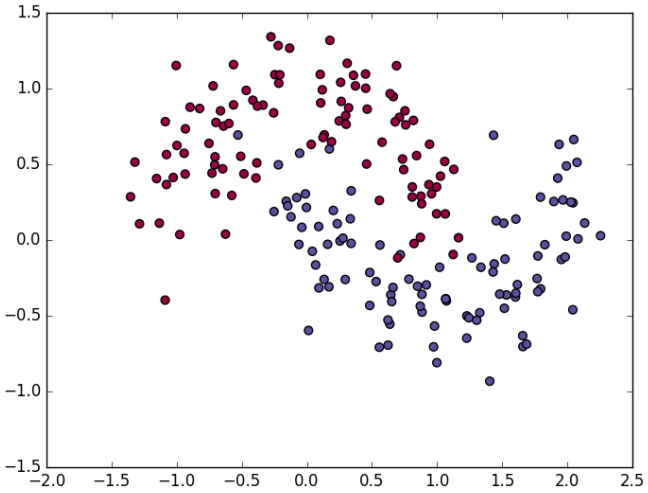

Figure 1: Two moons dataset displayed in matplotlib

1.2 Part B - Activation Functions

1.2.1 Part B1 - Implement activation functions

If you look in my code you can see I implemented the activations function.

1.2.2 Part B2 - Derivative Derivations

Part B2.1 - Derivative for Sigmoid function

$$
\begin{aligned}
\sigma(z) = f(z) = \frac{1}{1+e^{-z}} &= (1+e^{-z})^{-1} \\
\frac{df}{dz} := -1 * (1+e^{-z})^{-2} * -e^{-z} &= e^{-z}(1+e^{-z})^{-2} \quad \text{(by chain rule)} \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1-e^{-z}+1}{(1+e^{-z})^2} \quad \text{(add and subtract a 1)} \\
&= \frac{1+e^{-z}}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2} \quad \text{(split terms)} \\
&= \frac{1}{1+e^{-z}} - \frac{1}{(1+e^{-z})^2} \\
&= \sigma(z) - \frac{1}{(1+e^{-z})^2} \\
&= \sigma(z) - \left(\frac{1}{(1+e^{-z})}\right)^2 \\
&= \sigma(z) - \sigma(z)^2 \\
\boxed{f'(z) = \sigma(z)(1-\sigma(z))}
\end{aligned}
$$

Part B2.2 - Derivative for Tanh function

$$
\begin{aligned}
f(z) = \tanh(z) &= \frac{\sinh(z)}{\cosh(z)} \\
&= \frac{e^z - e^{-z}}{e^z + e^{-z}} \\
\frac{df}{dz} :&= \frac{\cosh(z) * \sinh'(z) - \sinh(z) * \cosh'(z)}{(\cosh(z))^2} \quad \text{(by quotient rule)} \\
&= \frac{\cosh(z)^2 - \sinh(z)^2}{(\cosh(z))^2} \\
&= \frac{\cosh(z)^2}{\cosh(z)^2} - \frac{\sinh(z)^2}{\cosh(z)^2} \\
\boxed{f'(z) = 1 - \tanh^2(z)}
\end{aligned}
$$

Part B2.3 - Derivative for ReLu function

$$
\begin{aligned}
\text{ReLu}(z) = f(z) &= \max(0, z) \\
\boxed{\frac{df}{dz} := \begin{cases} 0 & z \leq 0 \\ 1 & z \geq 0 \end{cases}} &\text{(I think the dirivitive here is pretty obvious)}
\end{aligned}
$$

1.2.3 Part B3 - Implement activation function gradiants

If you look in my code you can see I implemented the activations function gradiants.

1.3 Part C - Build the Neural Network

If you look in my code you can see I implemented the feedforward and loss functions

1.4 Part D - Backpropogation Derivations

1.4.1 Derivations of Backpropogation Equations

Keep in mind that the process for a single feedforward pass of the network is defined by the following equations with $x \in \Re^p$ being a single data sample with p features, $\mathbf{W_1} \in \Re^{n_1 \times p}$ is a weight matrix transitioning from the input later with p features to the number of units in hidden layer 1 n_1, $b_1 \in \Re^{n_1}$ is a bias vector, $z_1 \in \Re^{n_1}$ are a vector of potentials for layer 1, $a_1 \in \Re^{n_1}$ the corresponding activations, and $\hat{y}$ are the resulting probabilities. $\mathbf{W_2} \in \Re^{n_2 \times n_1}$, $a_1 \in \Re^{n_1}$, $z_2 \in \Re^{n_2}$

$$
\begin{aligned}
z_1 &= \mathbf{W_1} x + b_1 \\
a_1 &= \text{actFun}(z_1) \\
z_2 &= \mathbf{W_2} a_1 + b_2 \\
a_2 = \hat{y} &= \text{softmax}(z_2)
\end{aligned}
$$

Where the softmax function is given by:

$$\text{softmax}(\mathbf{z})_c = \frac{e^{z_c}}{\sum_{d=1}^{C} e^{z_d}} = \frac{e^{z_c}}{\Sigma_C} \quad \text{for } c = 1 \cdots C$$

With a cross entropy loss function of :

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{n \in N} \sum_{j \in C} y_{n,j} \log(\hat{y}_{n,j})$$

With y being a one hot vector of the correct label, N being the number of training examples and C being the number of classes.

For the following derivations, the derivative of the softmax $\frac{\partial \hat{y}_i}{\partial z_j}$ is useful to know. It is calculated below. It is the derivative of $\hat{y}$ with respect to one of the elements of the input vector z.

$$\hat{y} = \text{softmax}(\mathbf{z})_c = \frac{e^{z_c}}{\sum_{d=1}^{C} e^{z_d}} = \frac{e^{z_c}}{\Sigma_C} \quad \text{for } c = 1 \cdots C$$

$$\begin{aligned}
\text{if } i = j: \quad \frac{\partial y_i}{\partial z_i} &= \frac{\partial \frac{e^{z_i}}{\Sigma_C}}{\partial z_i} \\
&= \frac{e^{z_i}\Sigma_C - e^{z_i}e^{z_i}}{\Sigma_C^2} \\
&= \frac{e^{z_i}}{\Sigma_C}\frac{\Sigma_C - e^{z_i}}{\Sigma_C} \\
&= \frac{e^{z_i}}{\Sigma_C}(1 - \frac{e^{z_i}}{\Sigma_C}) \\
&= y_i(1 - y_i) \\
\text{if } i \neq j: \quad \frac{\partial y_i}{\partial z_j} &= \frac{\partial \frac{e^{z_i}}{\Sigma_C}}{\partial z_j} \\
&= \frac{0 - e^{z_i}e^{z_j}}{\Sigma_C^2} \\
&= -\frac{e^{z_i}}{\Sigma_C}\frac{e^{z_j}}{\Sigma_C} \\
&= -y_i y_j
\end{aligned}$$

The derivative of the cross entropy loss with respect to the second layer inputs are also a useful quantity to calculate, it is derived as follows. Note that I used the cross entropy for one example. Taking out the summation here makes things easier but I'll add it back in later.

$$\begin{aligned}
\frac{\partial L}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial z_{2_i}} = \frac{\partial L}{\partial z_{2_i}} &= -\sum_{j=1}^{C} \frac{\partial y_j log(\hat{y}_j)}{\partial z_i} \\
&= -\sum_{j=1}^{C} y_j \frac{\partial log(\hat{y}_j)}{\partial z_i} \\
&= -\sum_{j=1}^{C} y_j \frac{1}{\hat{y}_j}\frac{\partial \hat{y}_j}{\partial z_i} \\
&= -\frac{y_i}{\hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_i} - \sum_{j \neq i}^{C} \frac{y_j}{\hat{y}_j}\frac{\partial \hat{y}_j}{\partial z_i} \quad \text{substitution from above derivation} \\
&= -\frac{y_i}{\hat{y}_i}\hat{y}_i(1 - \hat{y}_i) - \sum_{j \neq i}^{C} \frac{y_j}{\hat{y}_j}(-\hat{y}_j\hat{y}_i) \\
&= -y_i + y_i\hat{y}_i + \sum_{j \neq i}^{C} y_j\hat{y}_i \\
&= -y_i + \sum_{j=1}^{C} y_j\hat{y}_i \\
&= -y_i + \hat{y}_i\sum_{j=1}^{C} y_j \\
&= \hat{y}_i - y_i
\end{aligned}$$

$\frac{dL}{dW_2}$ **Derivation** We can rewrite the loss function with substituting in the feed-forward equations above as follows.

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{n \in N} \sum_{j \in C} y_{n,j} \log(\hat{y}_{n,j})$$

$$\frac{\partial L}{\partial W_2} := \frac{\partial L}{\partial \hat{y}_{n,j}} \frac{\partial \hat{y}_{n,j}}{\partial z_2} \frac{\partial z_2}{\partial W_2}$$

Since j-th element of z_2 is given by:

$$z_{2_j} = \sum_{k=0}^{n_1} w_{j,k} a_{1_k} + b_{2_j}$$

The derivitive of the j-th element of z_2 w.r.t. weight $w_{j,k}$ is then:

$$\frac{\partial z_{2_j}}{\partial w_{j,k}} = a_{1_k}$$

Using the derivation above for $\frac{\partial L}{\partial z_{2_j}}$

$$\frac{\partial L}{\partial w_{2_{j,k}}} = (\hat{y}_j - y_j) a_{1_k}$$

$$\therefore$$

$$\frac{dL}{dW_2} = (\hat{\mathbf{y}} - \mathbf{y}) \mathbf{a_1^T} \in \Re^{n_2 \times n_1}$$

The above equation only takes into consideration on data point

We extend this to multiple samples by plugging in the summation we originally took out.

$$\boxed{\frac{dL}{dW_2} = \frac{1}{N} \sum_{n \in N} (\hat{\mathbf{y}}_\mathbf{n} - \mathbf{y}_\mathbf{n}) \mathbf{a_1^T} \in \Re^{n_2 \times n_1}}$$

$\frac{dL}{db_2}$ **Derivation** This derivation will be fairly similar to the previous section except now the gradient looks as follows:

$$\frac{\partial L}{\partial b_2} := \frac{\partial L}{\partial \hat{y}_{n,j}} \frac{\partial \hat{y}_{n,j}}{\partial z_2} \frac{\partial z_2}{\partial b_2}$$

Since j-th element of z_2 is given by:

$$z_{2_j} = \sum_{k=0}^{n_1} w_{j,k} a_{1_k} + b_{2_j}$$

The derivitive of the j-th element of z_2 w.r.t. bias b_{2_k} is then:

$$\frac{\partial z_{2_j}}{\partial b_{2_k}} = 1$$

Using the derivation above for $\frac{\partial L}{\partial z_{2_j}}$

$$\frac{\partial L}{\partial b_{2_k}} = (\hat{y}_j - y_j)$$

$$\therefore$$

$$\frac{dL}{db_2} = \hat{\mathbf{y}} - \mathbf{y} \in \Re^{n_2}$$

The above equation only takes into consideration on data point.
We extend this to multiple samples by plugging in the summation we originally took out.

$$\boxed{\frac{dL}{db_2} = \frac{1}{N} \sum_{n \in N} \hat{\mathbf{y}}_\mathbf{n} - \mathbf{y}_\mathbf{n} \in \Re^{n_2}}$$

$\frac{dL}{dW_1}$ **Derivation** We add one more layer to our previous derivation of $\frac{dL}{dW_2}$ for this derivation. The derivative can be expanded as follows

$$\frac{\partial L}{\partial W_1} := \frac{\partial L}{\partial \hat{y}_{n,j}} \frac{\partial \hat{y}_{n,j}}{\partial z_2} \frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial W_1}$$

So we need to figure out the term $\frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial W_1}$ by piecing to out. The j-th element of z_1 is given by:

$$z_{1_j} = \sum_{i=0}^{p} w_{1_{j,i}} x_i + b_{1_j}$$

So the derivative w.r.t. $w_{1_{j,i}}$ is then:

$$\frac{\partial z_{1_j}}{\partial w_{1_{j,i}}} = x_i$$

We know from part b that $\frac{\partial a_1}{\partial z_1}$ is dependent on the type of activation function but we derived all the derivatives of the three activation functions above so stating the derivatives is redundant. See the above part B.

$$\frac{\partial a_1}{\partial z_1} = \text{actFun}'(z_1) \quad \text{(See part B above)}$$

The last part we need to derive is the $\frac{\partial z_2}{\partial a_1}$ term.

Since the j-th element of z_2 is given by:

$$z_{2_j} = \sum_{k=0}^{n_1} w_{2_{j,k}} a_{1_k} + b_{2_j}$$

we can conclude that the derivative w.r.t. a_{1_k} is given by

$$\frac{\partial z_{2_j}}{\partial a_{1_k}} = w_{2_{j,k}}$$

From above, we know that the other partial derivatives are

$$\frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_{2_i}} = \frac{\partial L}{\partial z_{2_i}} = \hat{y}_i - y_i$$

So if we piece this all together we come up with:

$$\frac{\partial L}{\partial w_{1_{k,p}}} := \frac{\partial L}{\partial \hat{y}_{n,c}} \frac{\partial \hat{y}_{n,c}}{\partial z_{2_c}} \frac{\partial z_{2_c}}{\partial a_{1_k}} \frac{\partial a_{1_k}}{\partial z_{1_k}} \frac{\partial z_{1_k}}{\partial w_{1_{k,p}}}$$

Which I changed some of the indexing to make the most sense. In this format $n \in N$, $c \in C$, , $k \in \{0, ..., n_1 - 1\}$, $p \in \{0, ..., P - 1\}$ with P input features. The resulting derivative with respect to one weight in the first layer $w_{1_{k,p}}$ is then:

$$\frac{\partial L}{\partial w_{1_{k,p}}} = \sum_{c \in C} (\hat{y}_c - y_c) w_{2_{c,k}} \text{actFun}'(z_{1_k}) x_p$$

and the extention to matrix form is then

$$\frac{\partial L}{\partial \mathbf{W_1}} = (\hat{\mathbf{y}} - \mathbf{y}) \mathbf{W_2} \odot \text{actFun}'(\mathbf{z_1}) \mathbf{x^T} \in \Re^{n_1 \times p}$$

If there is morethan one sample (which there always is)
then we averageover all of the samples and the final solution is then

$$\boxed{\frac{\partial L}{\partial \mathbf{W_1}} = \frac{1}{N} \sum_{n \in N} (\hat{\mathbf{y}} - \mathbf{y}) \mathbf{W_2} \odot \text{actFun}'(\mathbf{z_1}) \mathbf{x^T} \in \Re^{n_1 \times p}}$$

$\frac{dL}{db_1}$ **Derivation** The derivation is very similar to the $\frac{dL}{dW_1}$ except the last part of the chain of partial derivatives is changed. The gradient equation of the Loss function with respect to a single bias term in the first layer is

$$\frac{\partial L}{\partial b_{1_k}} := \frac{\partial L}{\partial \hat{y}_{n,c}} \frac{\partial \hat{y}_{n,c}}{\partial z_{2_c}} \frac{\partial z_{2_c}}{\partial a_{1_k}} \frac{\partial a_{1_k}}{\partial z_{1_k}} \frac{\partial z_{1_k}}{\partial b_{1_k}}$$

So we only need to calculate $\frac{\partial z_{1_k}}{\partial b_{1_k}}$ first. Let's look at how the k th value is calculated.

$$z_{1_k} = \sum_{i=0}^{p} w_{1_{k,i}} x_i + b_{1_k}$$

So the derivative w.r.t. b_{1_k} is then:

$$\frac{\partial z_{1_j}}{\partial b_{1_k}} = 1$$

The partial derivative $\frac{\partial z_{1_k}}{\partial b_{1_k}} = 1$ simplified the equation for the partial derivative of the loss function w.r.t. b_{1_k} as

$$\frac{\partial L}{\partial b_{1_k}} = \sum_{c \in C} (\hat{y}_c - y_c) w_{2_{c,k}} \text{actFun}'(z_{1_k})$$

and the resulting matrix form is then

$$\frac{\partial L}{\partial \mathbf{b_1}} = (\hat{\mathbf{y}} - \mathbf{y}) \mathbf{W_2^T} \text{actFun}'(\mathbf{z_1}) \in \Re^{n_1}$$

If there is more than one sample (which there always is) then we have to sum up and average over all samples and the gradient becomes

$$\boxed{\frac{\partial L}{\partial \mathbf{b_1}} = \frac{1}{N} \sum_{n \in N} (\hat{\mathbf{y}} - \mathbf{y}) \mathbf{W_2^T} \odot \text{actFun}'(\mathbf{z_1}) \in \Re^{n_1}}$$

1.4.2 Implementation of Derivations of Backpropogation Equations

If you look in my code you can see I implemented the backpropogation equations correctly.

1.5 Time to Have Fun - Training!

1.5.1 Change the activation functions and number of hidden units Part 1 and 2

The decision boundaries are shown for tanh, sigmoid, and relu functions for 3, 10 and 20 hidden hidden units. Using the default 3 hidden units with different activation functions, we get Figures 2,3, and 4.

Increasing the number of hidden units to 10 gives us figures 5, 6 and 7

Increasing the number of hidden units to 20 gives us figure 8, 9 and 10

The Tanh activation function seems to be the most expressive, it curves much more than the sigmoid and relu activations functions. The Relu activation function decision boundary looks piecewise linear which makes sense because it is. The sigmoid function looks practically linear for a small number of hidden units, but as the number of hidden units is increased, we seem more expressively in the decision boundary.

As we increase the number of hidden units in the Tanh network specifically, the decision boundary becomes very expressive and using 20 hidden units, is almost able to create a decision boundary that separates the two classess.

1.6 Building a deeper network

In my github I have created the n layer neural network. py file where I have implemented another class called deep network which inherits from the three layer network. I pretty much followed the hints provided in the homework to created the deep neural network.

I have tested the neural network with various 1,2,3,4, and 5 layer networks with various activation functions and sizes for layers. It generally helps that the layers have a decreasing number of hidden units as you feedforward I have found out. A 5 layer network with decreaseing numbers of hidden units 20,16,10,4,4 outperforms a 5 layer network with all the same 10,10,10,10,10 and also better than and network with an increasing number of hidden units 4,4,10,16,20 which makes intuitive sense. The tanh activation function also seems to perform the best. Figures 11, 12 and 13 show decision boundaries after 10000 iterations of 2 layer network with different activation functions.

Figures 14, 15 and 16 show decision boundaries after 10000 iterations of 3 layer network with different activation functions. Sigmoids just suck.

Figures 17 and 18 show decision boundaries after 10000 iterations of 4 layer network with the tanh and relu activation functions. After this point sigmoids still do not perform well and I didn't even include the plot.

It seems to be a general trend that as you go deeper, you get more interesting decision boundaries, the network has more expressivity, and loss goes down, which are all great things.

1.6.1 Use a different dataset

The different dataset I choose to use was the 'make_circles' dataset. I choose this dataset because it was the first dataset on the website that I could find and it is also 2 dimensional to not complicate the problem. I experimented with different networks with varying amounts of layers, activation function. One of the interesting things that I found is that some of the networks are very finicky. I had a 5 layer [20,16,10,4,4] hidden unit network that I was using on the two circles dataset that worked really well when the noise was around 15% shown in figure 19 but when I increased the noise just a little bit the network fell apart as shown by the decision boundary in figure 20. I experimented with various architectures and different activation functions like in the previous section but I found the 5 layer [20,16,10,4,4] hidden unit network with a tanh activation function worked the best.

2 Training a Simple Deep Convolutional Network on MNIST

Parts 1-4 require use to read documents and fill in code, you can see I filled in the code from my github. Part 4 asks what the final training accuracy is after the network is run. My test accuracy is 98.78%.

2.1 Visualize the training

In this section we are asked to use tensorboard to visualize variables that are of importance to us during training. We can visualize the mean error as a function of the number of training iterations this way given by figure 21.

If we look at the graphs in the 'graphs' section of TensorBoard, we can see the variables in Tensorflow as well as how these variables interact with each other via the network architecture. The variables and network architecture are given by figure 22.

2.2 Visualize the training More!

After implementing the min, max, mean, standard deviation and histogram functions into the dcn-mnist.py file, we can now visualize these characteristics of the weights, biases, inputs, activations at the relu and activations at the max pooling for every layer. Since for every layer, this is around 25 plots, I will only provide the first 2 layers of these characteristics to show that I've implemented the feature. I also include errors for training, validation and testing. All of these plots are figures 23 thru 73

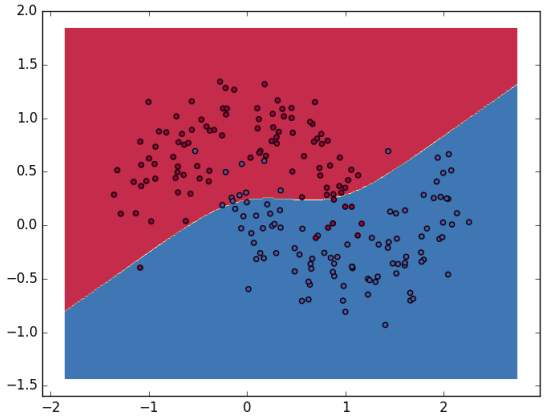

Figure 2: Default settings, 3 hidden units with tanh activation fuctions. Loss = .260

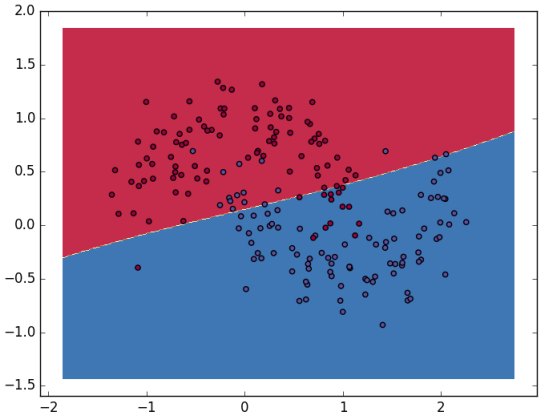

Figure 3: 3 hidden units with sigmoid activation fuctions. Loss = .304

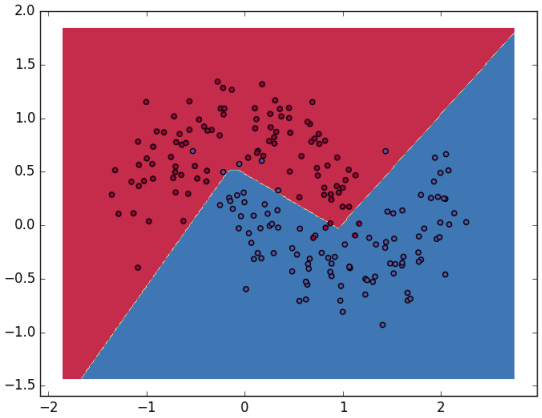

Figure 4: 3 hidden units with ReLU activation fuctions. Loss = .235

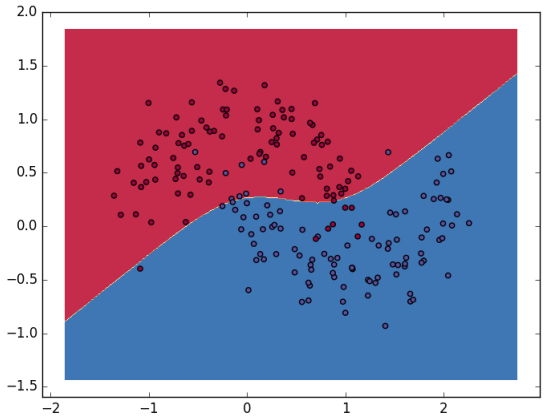

Figure 5: 10 hidden units with tanh activation fuctions. Loss = .246

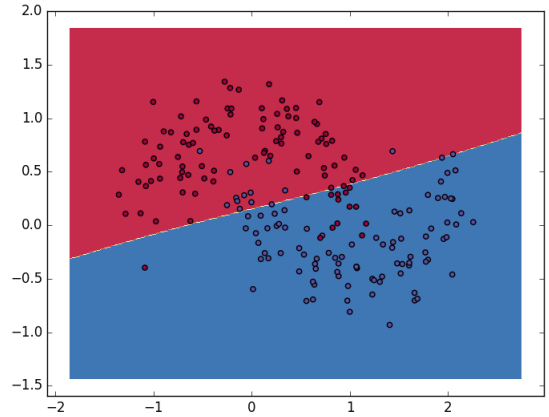

Figure 6: 10 hidden units with sigmoid activation fuctions. Loss = .301

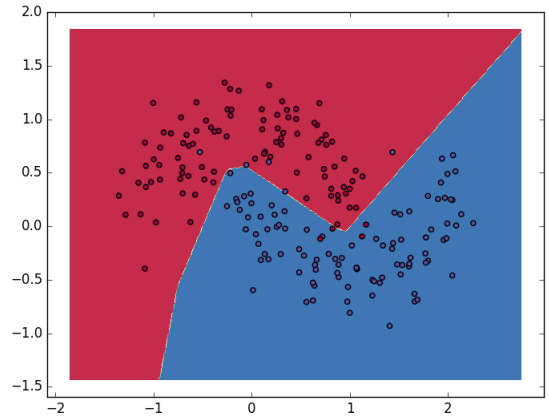

Figure 7: 10 hidden units with ReLU activation fuctions. Loss = .202

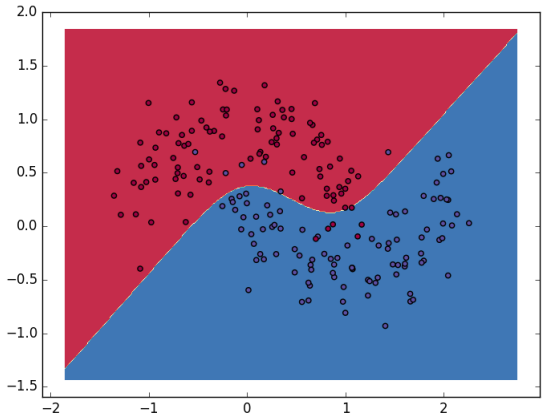

Figure 8: 20 hidden units with tanh activation fuctions. Loss = .191

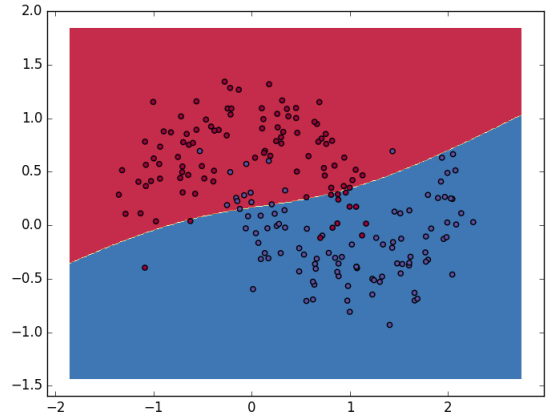

Figure 9: 20 hidden units with sigmoid activation fuctions. Loss = .288

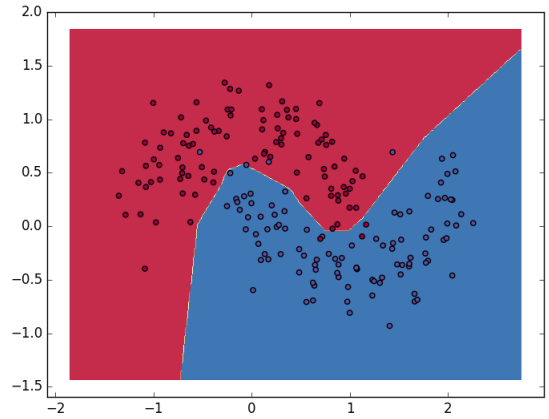

Figure 10: 20 hidden units with ReLU activation fuctions. Loss = .150

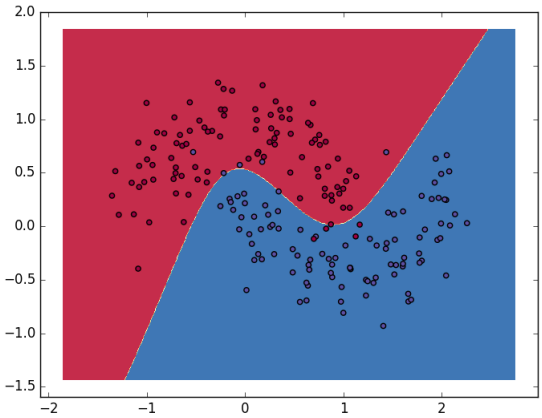

Figure 11: 2 layer network with hidden units [20,10] with a tanh activation fuction. Loss = .14

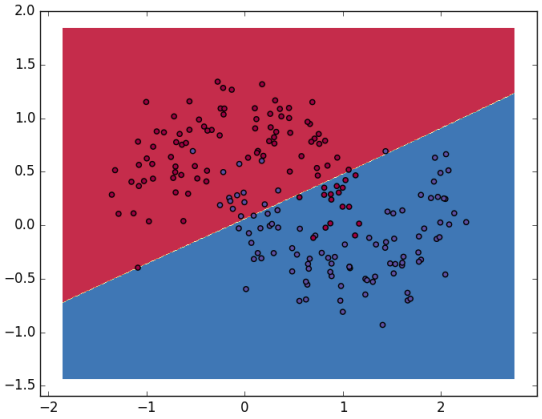

Figure 12: 2 layer network with hidden units [20,10] with a sigmoid activation fuction. Loss = .349

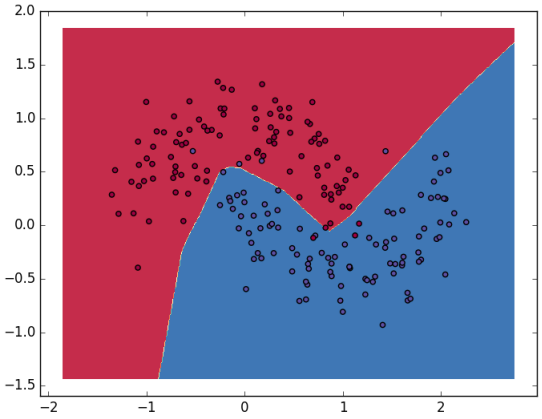

Figure 13: 2 layer network with hidden units [20,10] with a relu activation fuction. Loss = .142

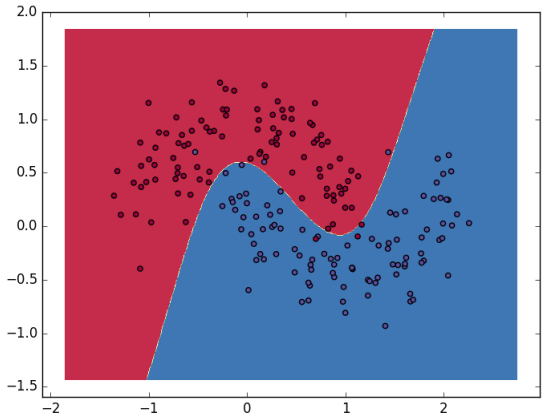

Figure 14: 3 layer network with hidden units [30,20,10] with a tanh activation fuction. Loss = .14

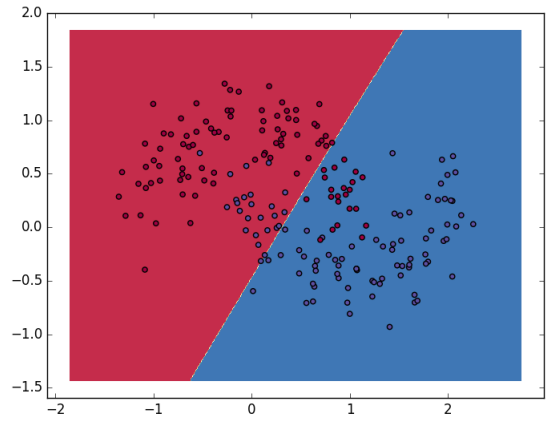

Figure 15: 3 layer network with hidden units [30,20,10] with a sigmoid activation fuction. Loss = .349

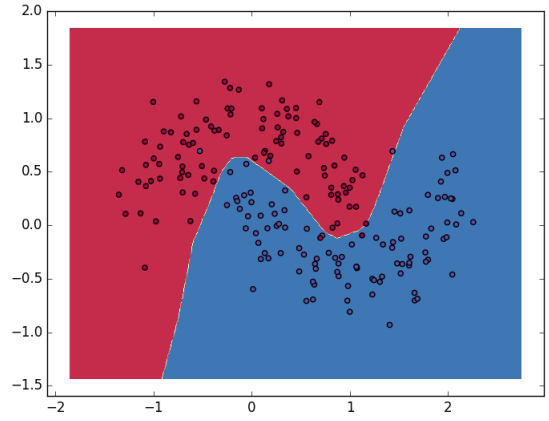

Figure 16: 3 layer network with hidden units [30,20,10] with a relu activation fuction. Loss = .142

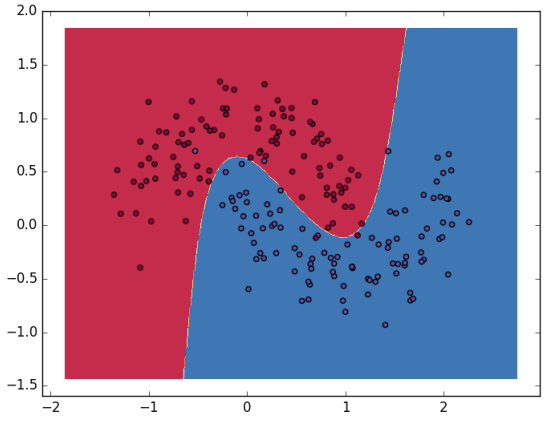

Figure 17: 4 layer network with hidden units [10,10,6,3] with a tanh activation fuction. Loss = .09

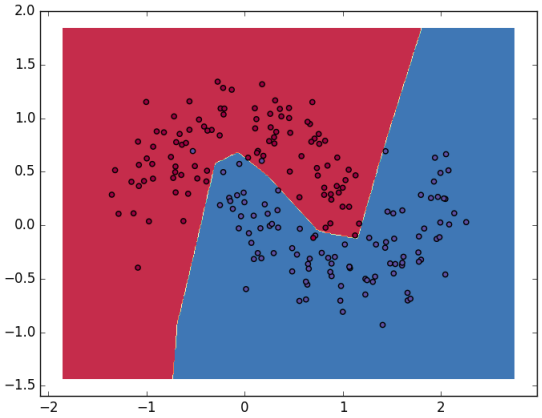

Figure 18: 4 layer network with hidden units[10,10,6,3] with a relu activation fuction. Loss = .142

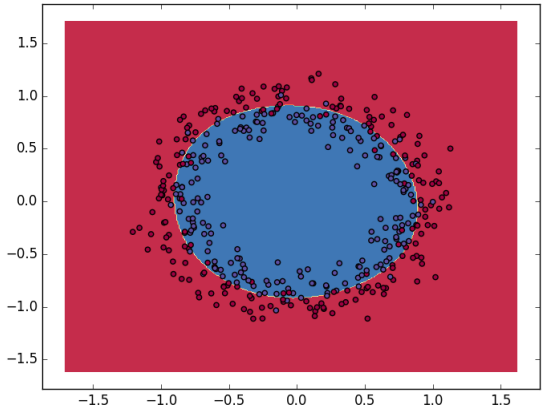

Figure 19: 5 layer [4,4,10,16,20] network with tanh activtion function on two cirlces dataset showing a nice decision boundary when noise is low. . Loss = .08

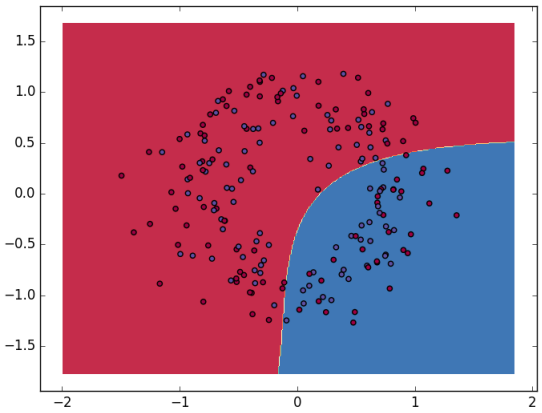

Figure 20: 5 layer [4,4,10,16,20] network with tanh activtion function on two cirlces dataset showing a bad decision boundary when noise is turned up by 5 percent. Loss = .69

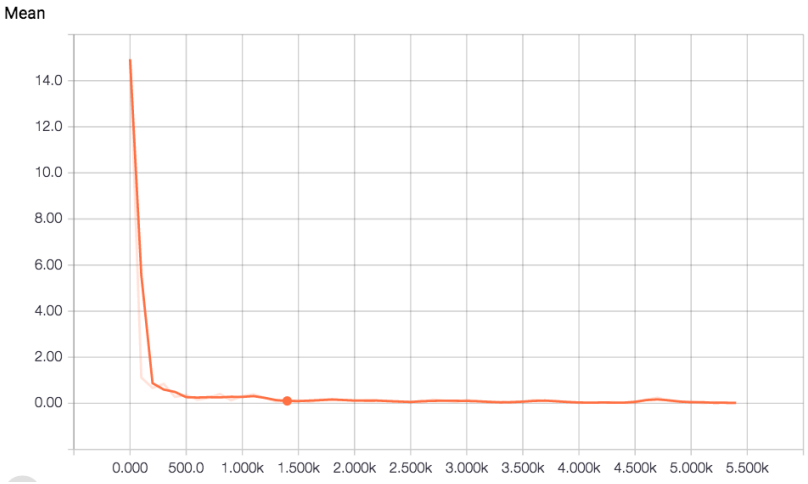

Figure 21: Mean Error as function of # of training iterations

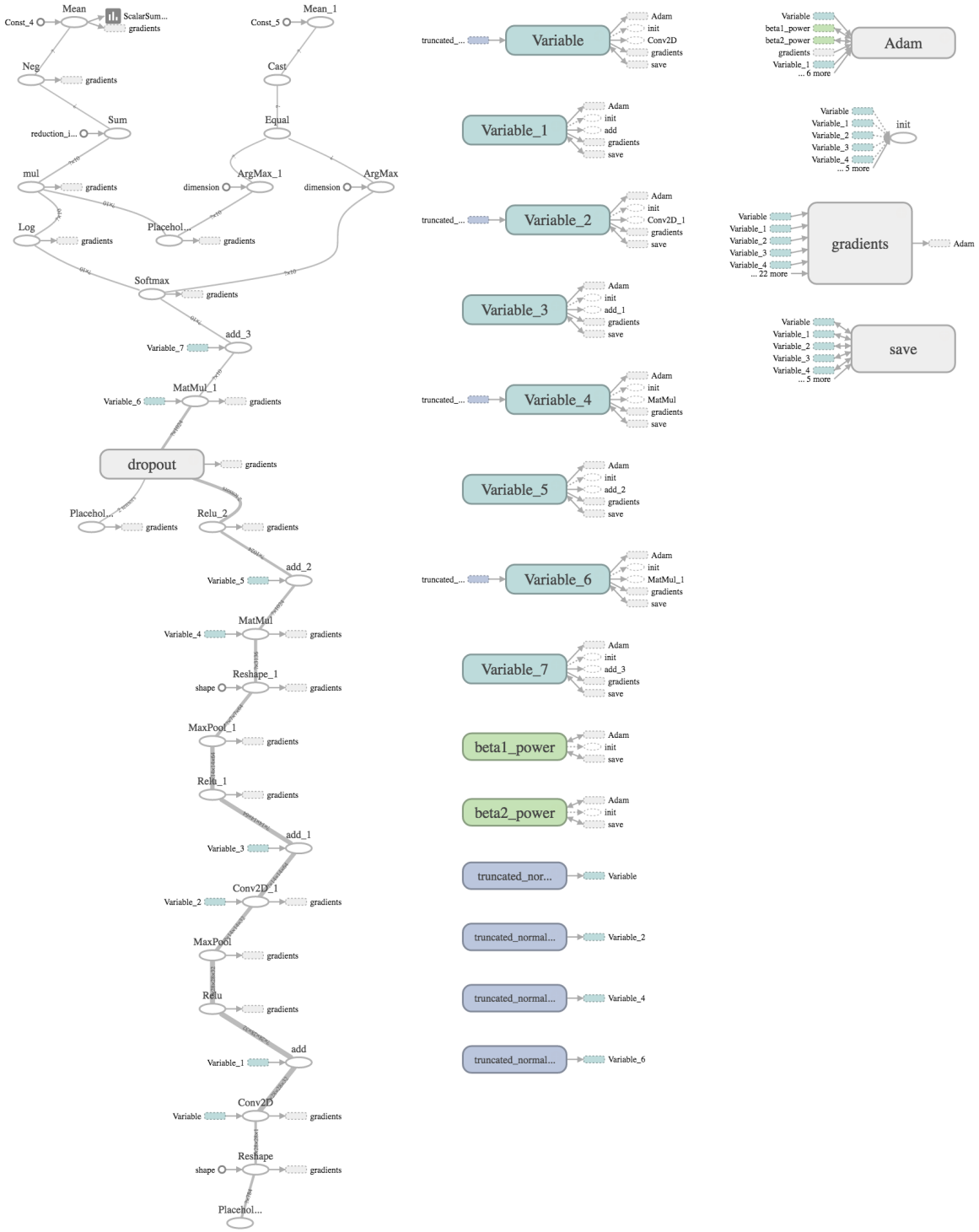

Figure 22: Network Architecture

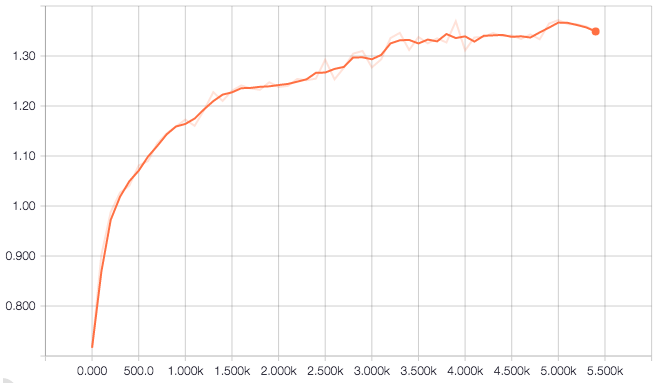

Figure 23: Plot of maximum of activations after relu in layer one for all iterations

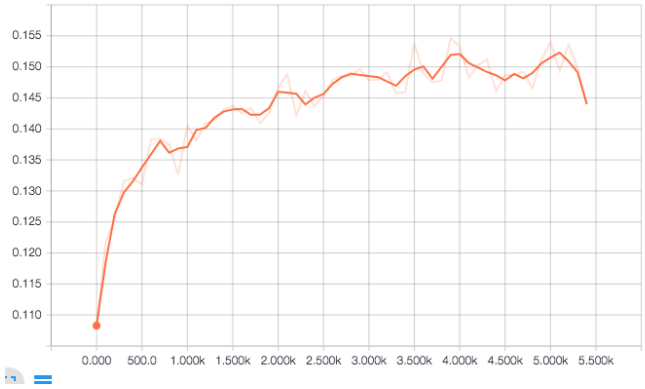

Figure 24: Plot of mean of activations after relu in layer one for all iterations

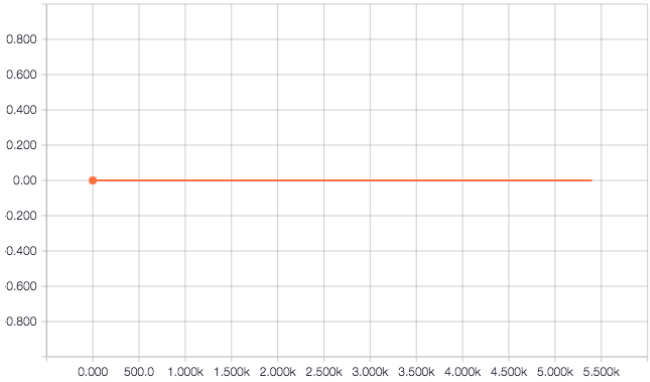

Figure 25: Plot of min of activations after relu in layer one for all iterations

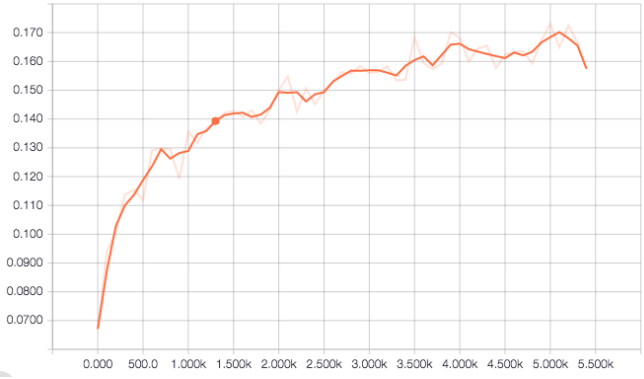

Figure 26: Plot of std of activations after relu in layer one for all iterations

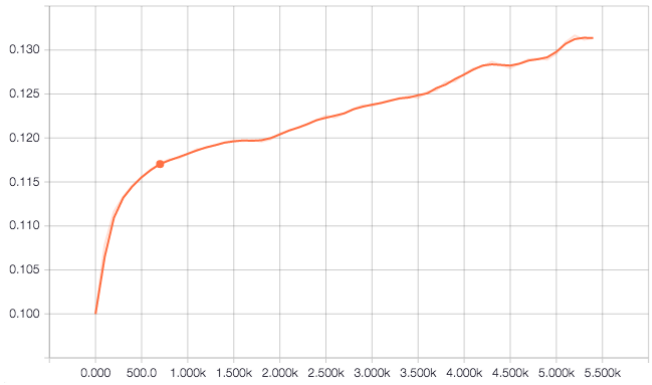

Figure 27: Plot of maximum of biases in layer one for all iterations

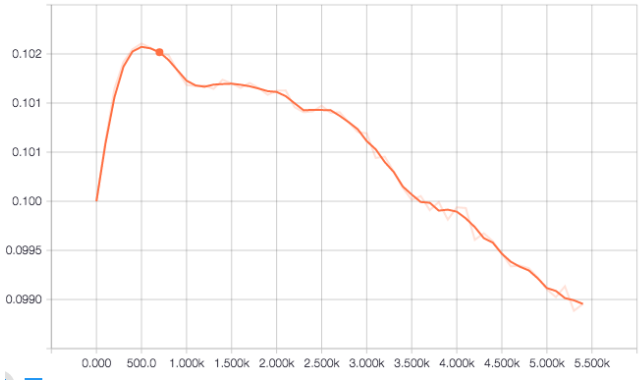

Figure 28: Plot of mean of biases in layer one for all iterations

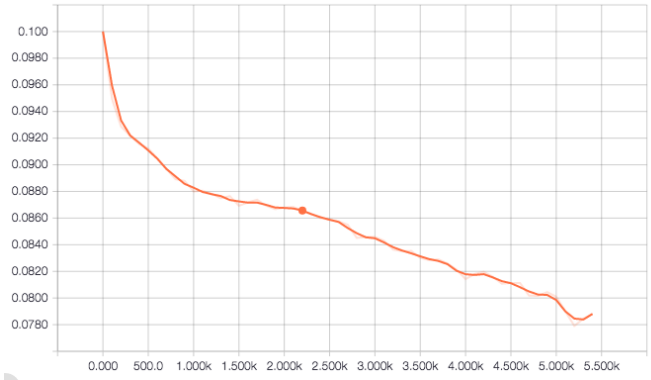

Figure 29: Plot of min of biases in layer one for all iterations

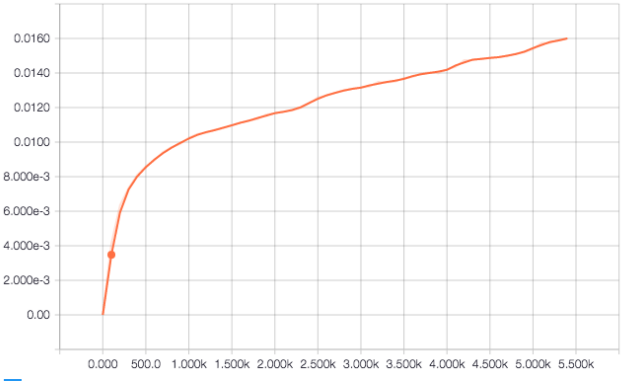

Figure 30: Plot of std of biases in layer one for all iterations

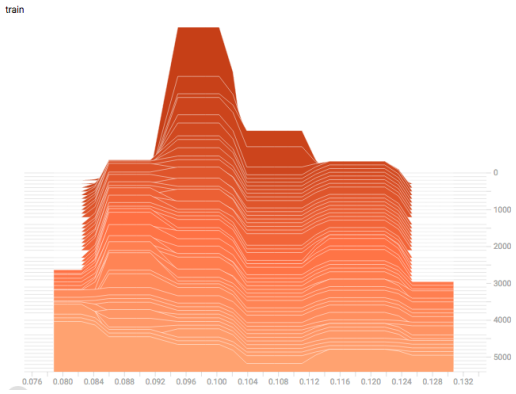

Figure 31: Histogram of biases in layer one for all iterations

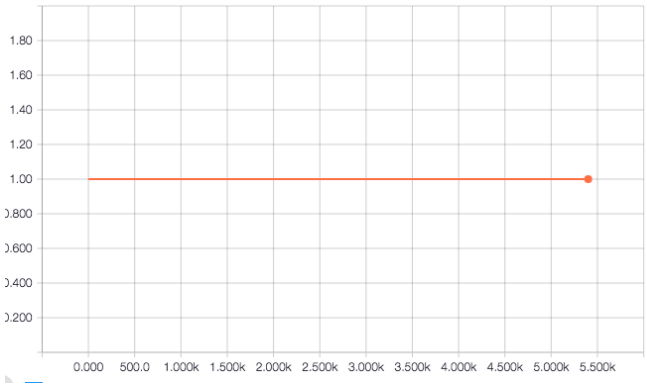

Figure 32: Plot of maximum of inputs to layer one for all iterations

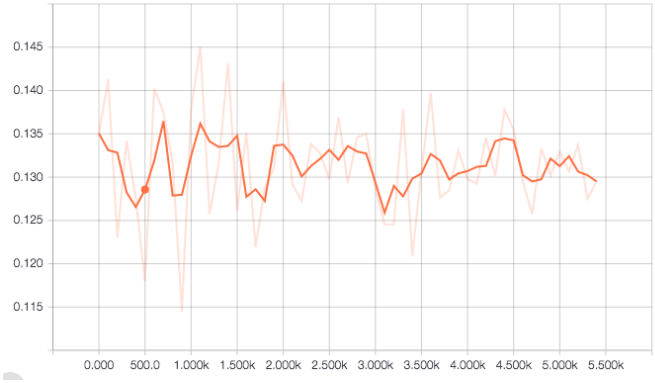

Figure 33: Plot of mean of inputs to layer one for all iterations

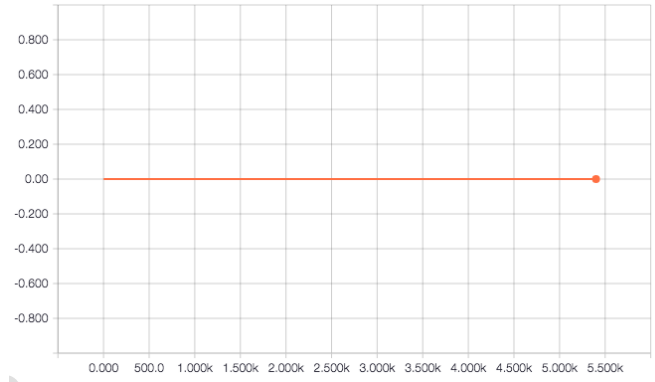

Figure 34: Plot of min of inputs tolayer one for all iterations

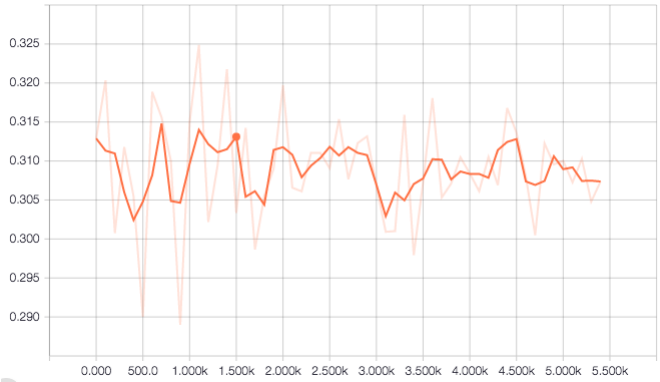

Figure 35: Plot of std of inputs tolayer one for all iterations

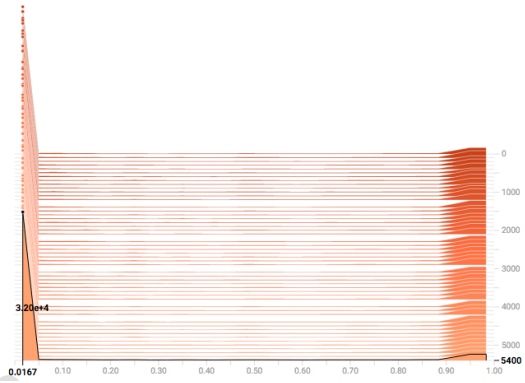

Figure 36: Histogram of inputs to layer one for all iterations

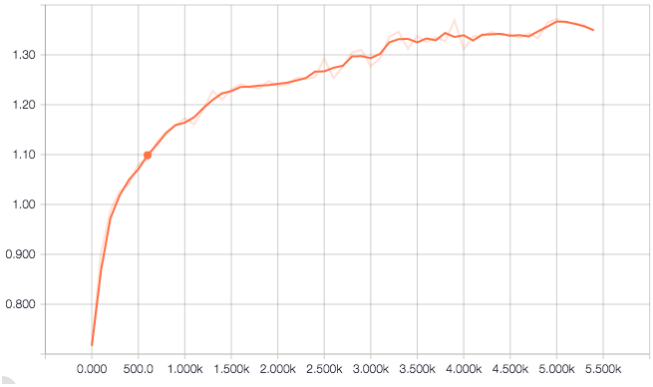

Figure 37: Plot of maximum of activtions after max pooling in layer one for all iterations

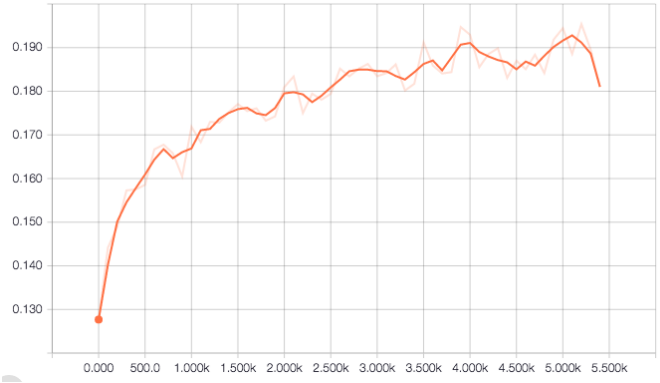

Figure 38: Plot of activtions after max pooling in inputs layer one for all iterations

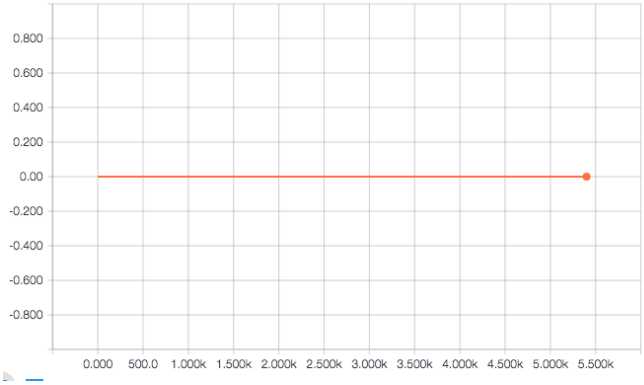

Figure 39: Plot of activtions after max pooling in layer one for all iterations

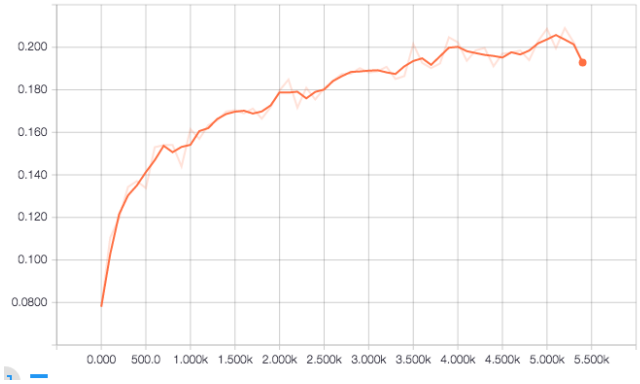

Figure 40: Plot of activtions after max pooling in layer one for all iterations

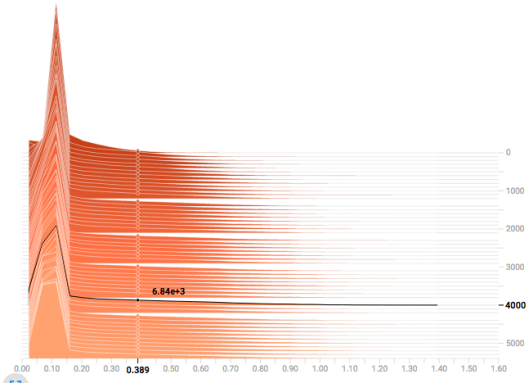

Figure 41: Histogram of activtions after max pooling in layer one for all iterations

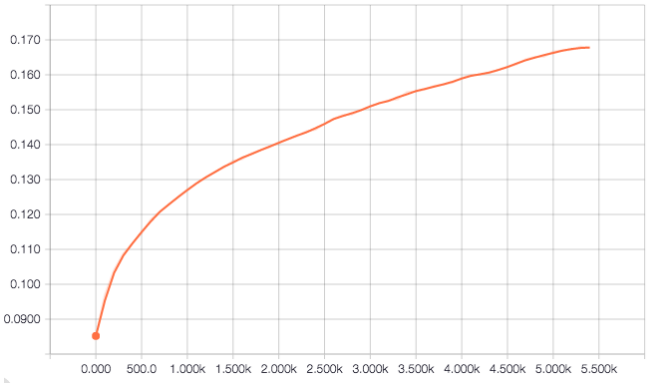

Figure 42: Plot of maximum of weights in layer one for all iterations

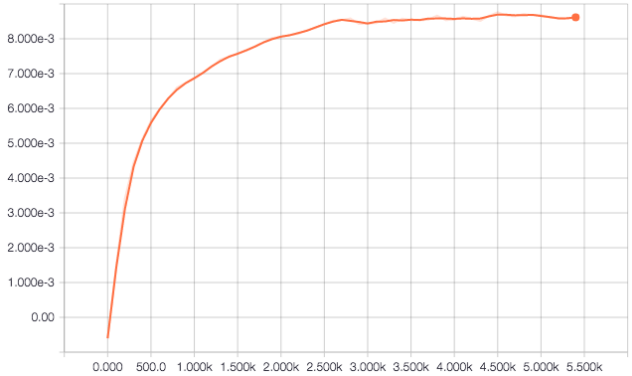

Figure 43: Plot of weights in inputs layer one for all iterations

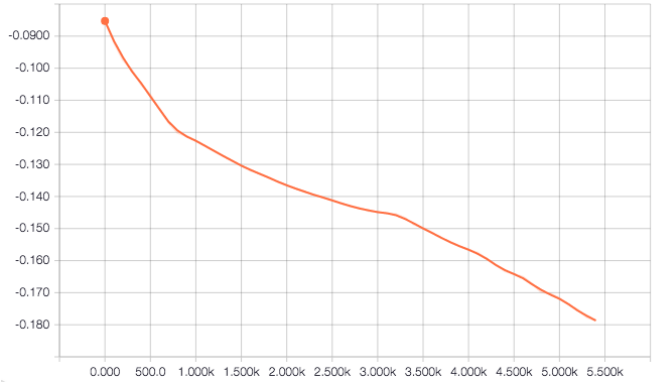

Figure 44: Plot of weights in layer one for all iterations

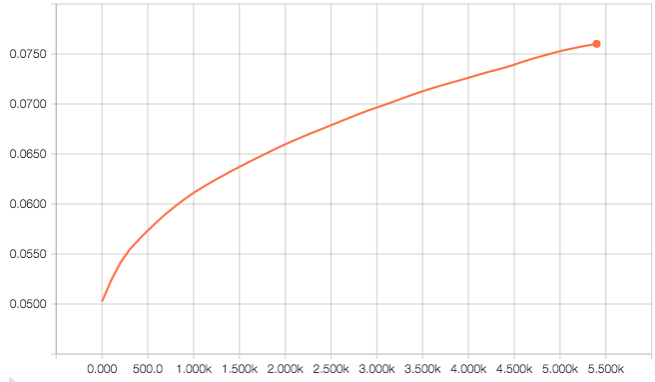

Figure 45: Plot of weights in layer one for all iterations

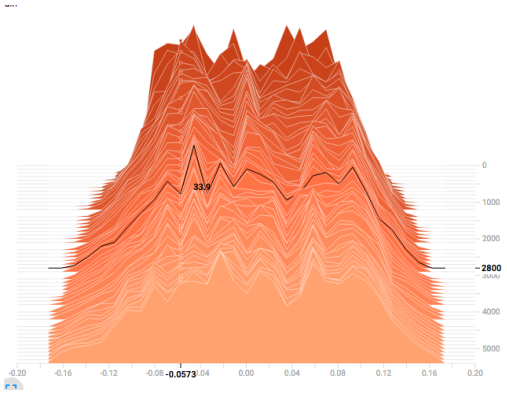

Figure 46: Histogram of weights in layer one for all iterations

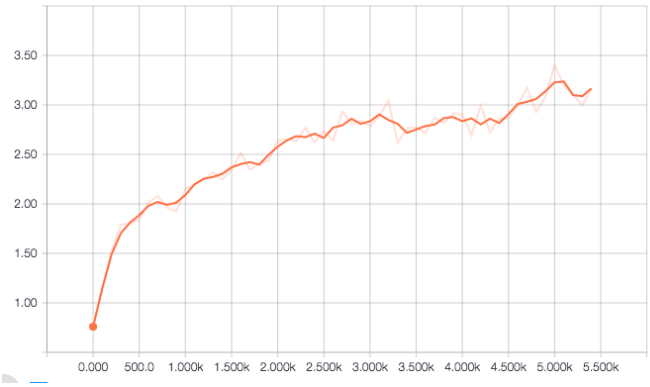

Figure 47: Plot of maximum of activations after relu in layer two for all iterations

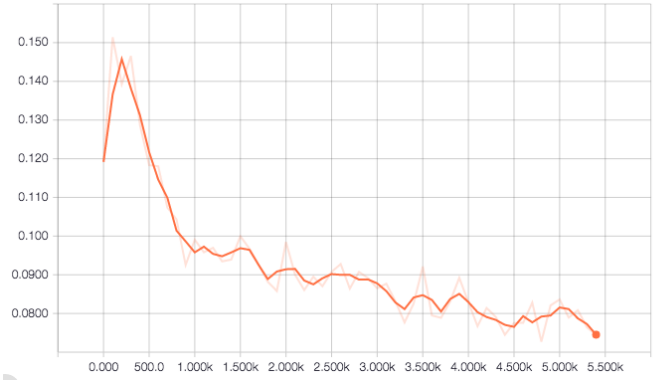

Figure 48: Plot of mean of activations after relu in layer two for all iterations

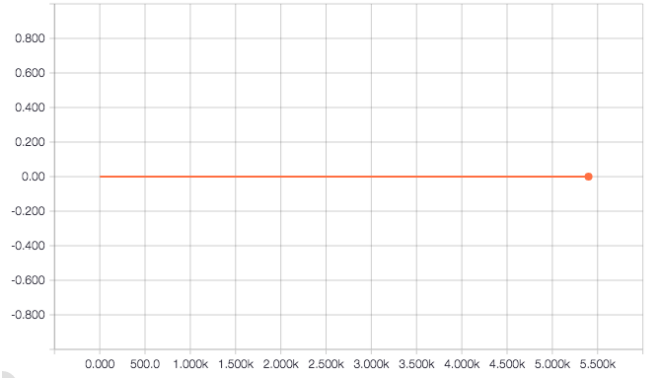

Figure 49: Plot of min of activations after relu in layer two for all iterations

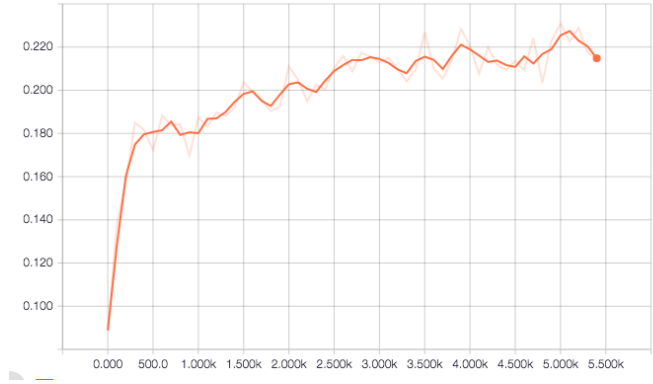

Figure 50: Plot of std of activations after relu in layer two for all iterations

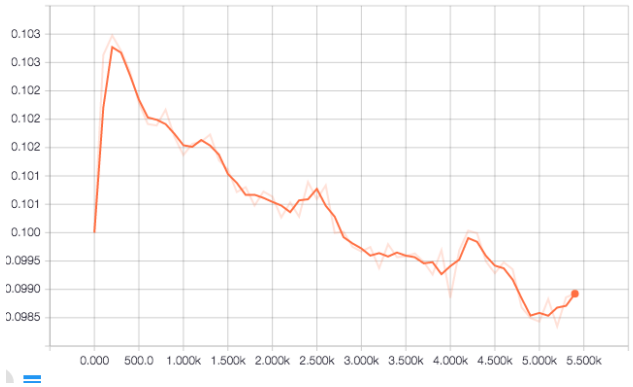

Figure 51: Plot of maximum of biases in layer one for all iterations

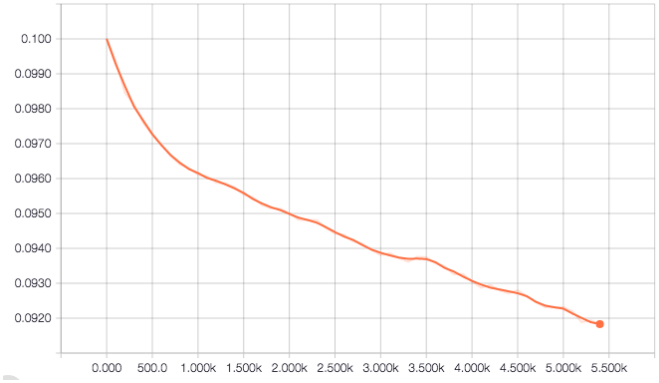

Figure 52: Plot of mean of biases in layer one for all iterations

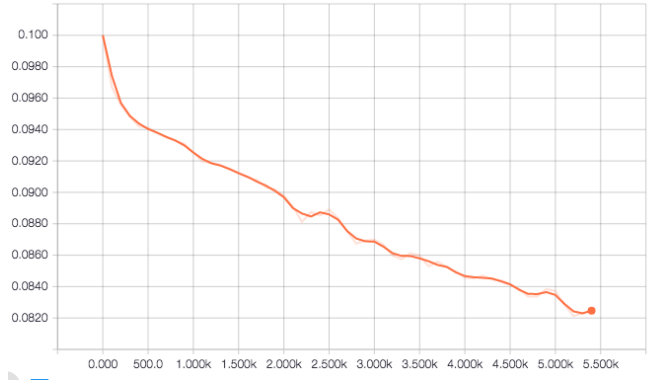

Figure 53: Plot of min of biases in layer one for all iterations

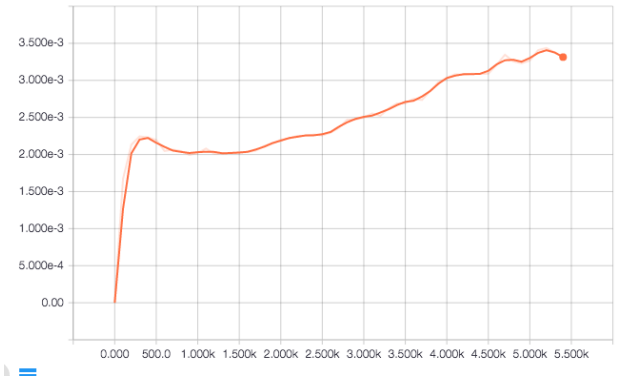

Figure 54: Plot of std of biases in layer one for all iterations

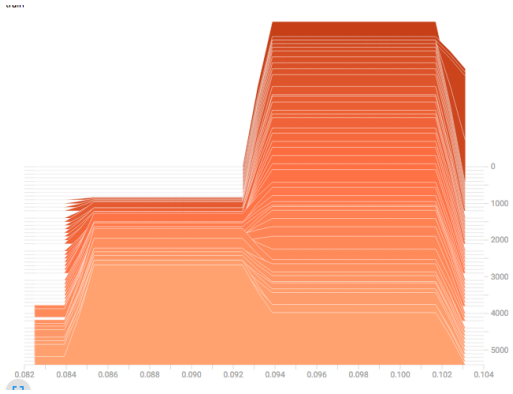

Figure 55: Histogram of biases in layer one for all iterations

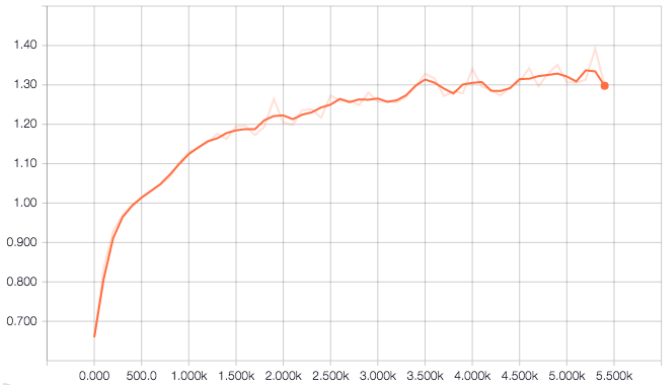

Figure 56: Plot of maximum of inputs to layer one for all iterations

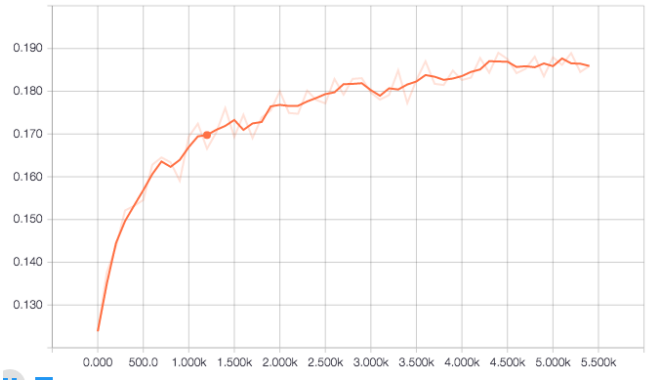

Figure 57: Plot of mean of inputs to layer one for all iterations

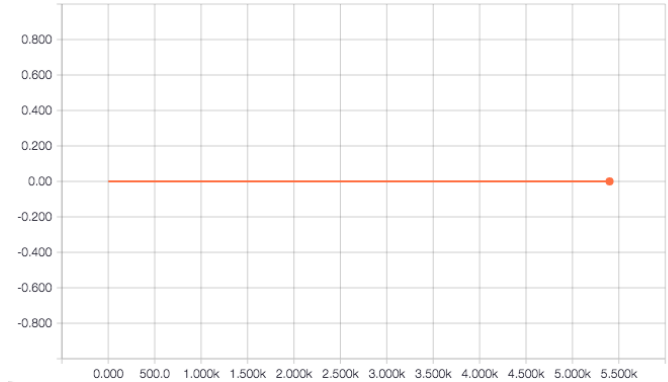

Figure 58: Plot of min of inputs tolayer one for all iterations

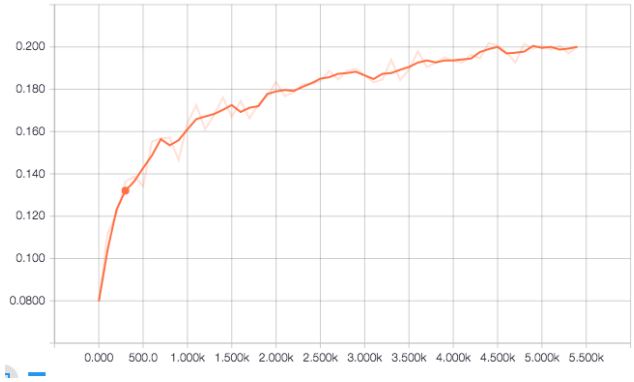

Figure 59: Plot of std of inputs tolayer one for all iterations

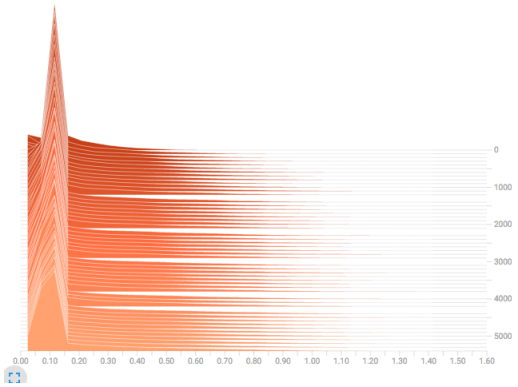

Figure 60: Histogram of inputs to layer one for all iterations

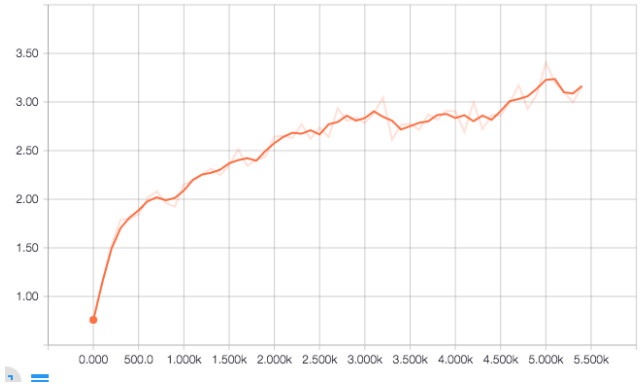

Figure 61: Plot of maximum of activtions after max pooling in layer one for all iterations

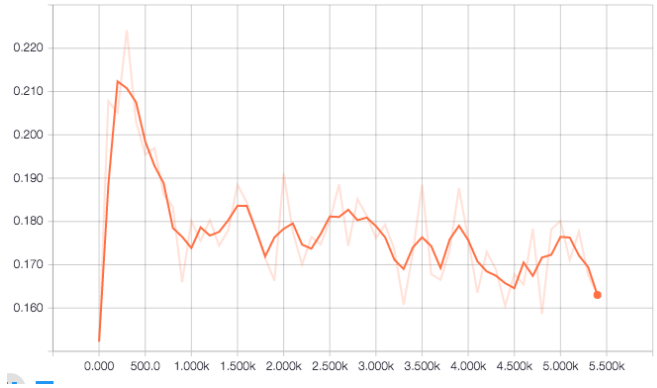

Figure 62: Plot of activtions after max pooling in inputs layer one for all iterations

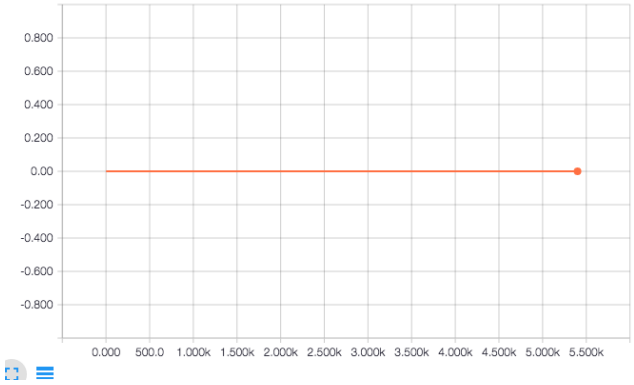

Figure 63: Plot of activtions after max pooling in layer one for all iterations

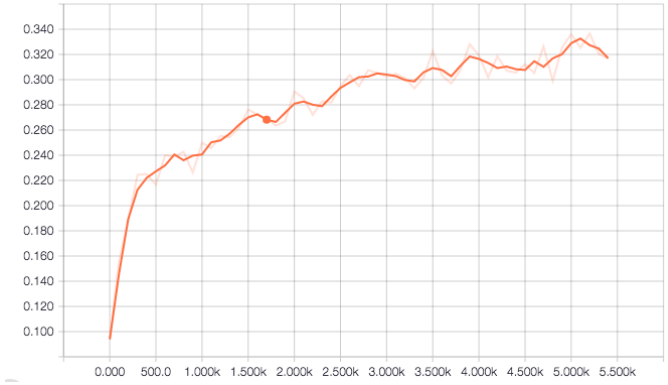

Figure 64: Plot of activtions after max pooling in layer one for all iterations

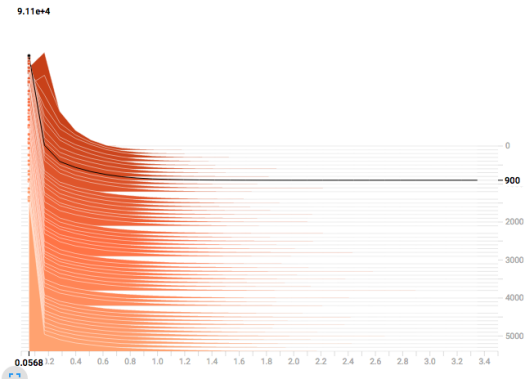

Figure 65: Histogram of activtions after max pooling in layer one for all iterations

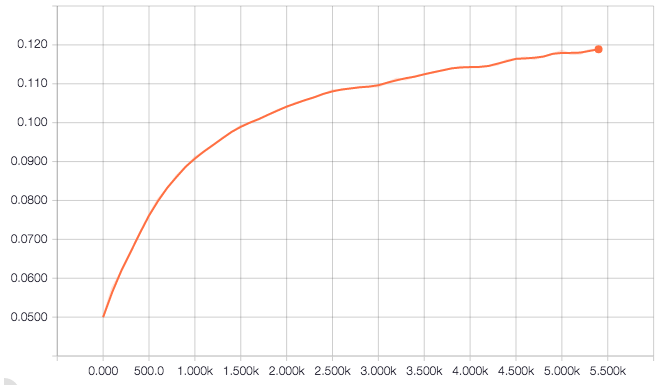

Figure 66: Plot of maximum of weights in layer one for all iterations

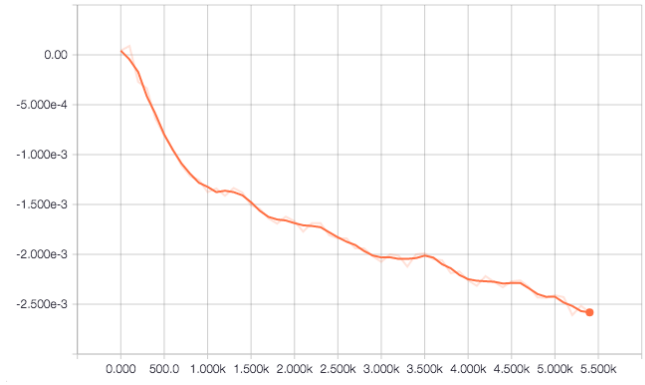

Figure 67: Plot of weights in inputs layer one for all iterations

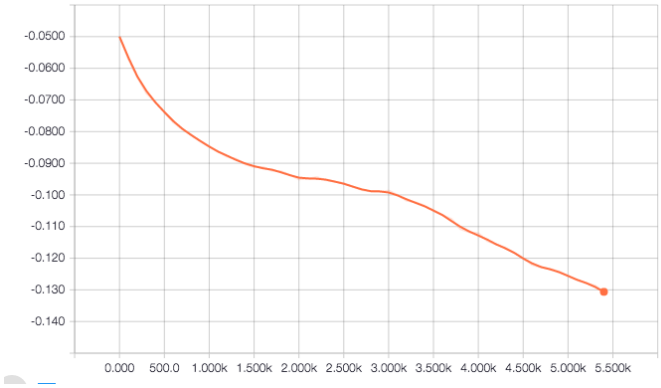

Figure 68: Plot of weights in layer one for all iterations

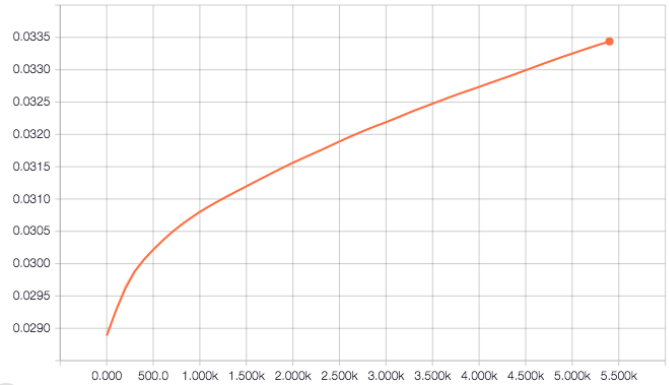

Figure 69: Plot of weights in layer one for all iterations

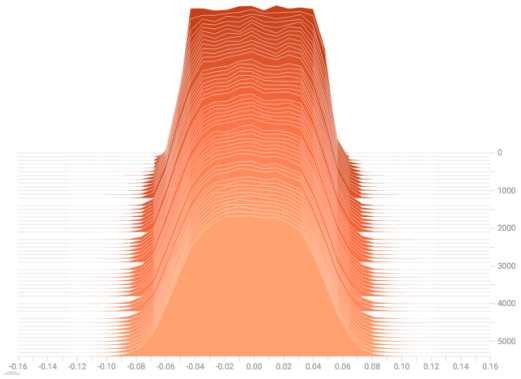

Figure 70: Histogram of weights in layer one for all iterations

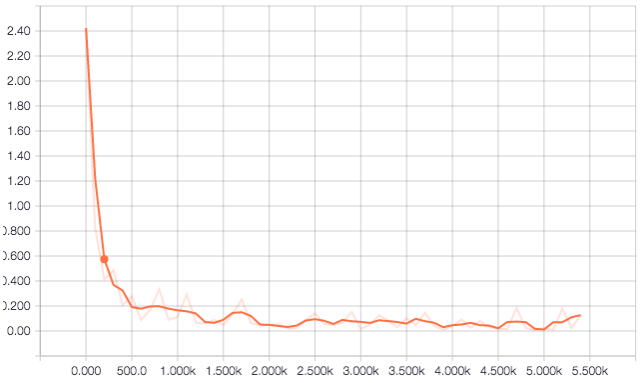

Figure 71: Training Error for the default covNet

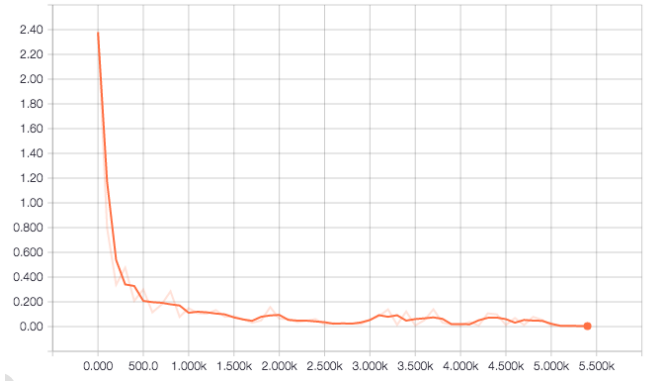

Figure 72: Validation Error for the default covNet

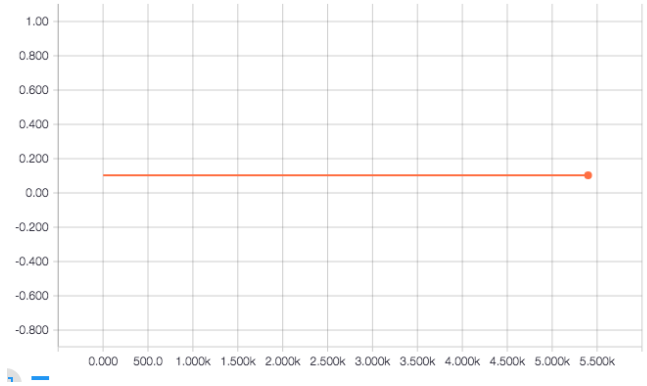

Figure 73: Testing Error for the default covNet

2.3 Visualize the training More!

This section of the homework asks us to try different activation functions, initializations, and training algorithms and display the results from Tensorboard. I will just provide the histograms of the important variables from the first and second layer becasue those histograms have the mean, min, max, and std of the variable inherently coded in them. I will show 2 different configurations to show that I've been experimenting but adding all of the figures the report is a tedious, time consuming process that takes way too much time than what it is worth. The first is using Xavier initialization, a ReLU non-linearity, and adaGrad in figures 74 thru 84 with a resulting testing error of 1.9%. The second configuration is using a leaky ReLu, Xavier initialization, and SGD and the histograms of the 5 variables in each layer are given by figures 85 thru 95 with a resulting testing error 2.2% error. Different combination of non-linearities, initializations, and optimization methods provide different results. I found that generally every network can be improved with Xavier initialization and in this case adaGrad helped out training a lot. But that may not be true for all scenarios and datasets. I found that pretty much all the fancy optimization methods like adaGrad, and momentum based methods did really well, beating SGD. But SGD is still very good becasue it is simple and pretty much always works. Overall I thought this was a very long homework but I learned a lot about training networks. I could do without the really large reports and plot making though.

3 Link to my code

`https://github.com/jjm7/Elec677/tree/Homework1`

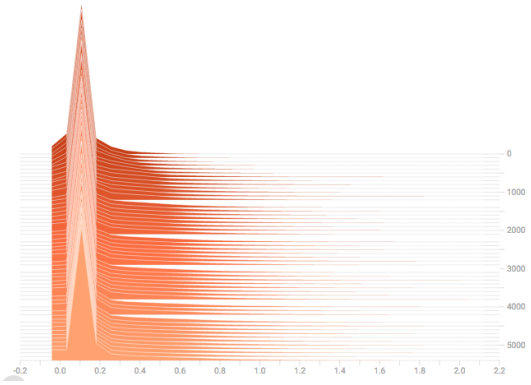

Figure 74: Histogram of activations in layer one for all iterations

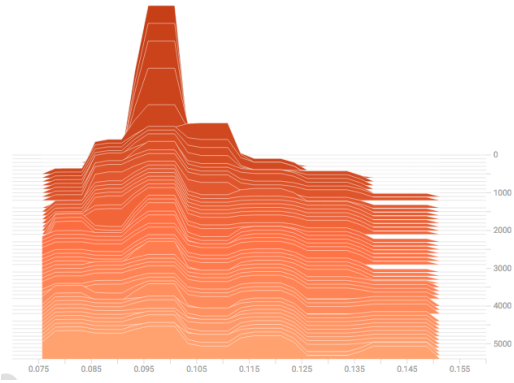

Figure 75: Histogram of biases in layer one for all iterations

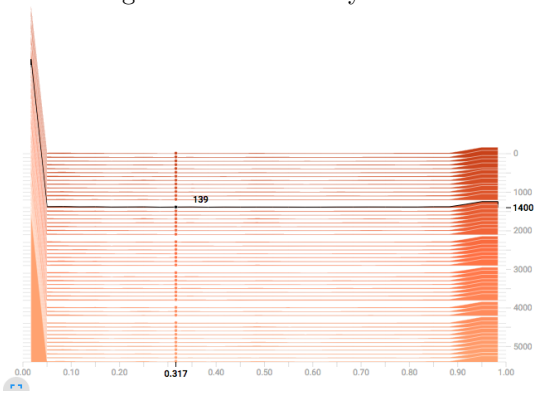

Figure 76: Histogram of inputs in layer one for all iterations

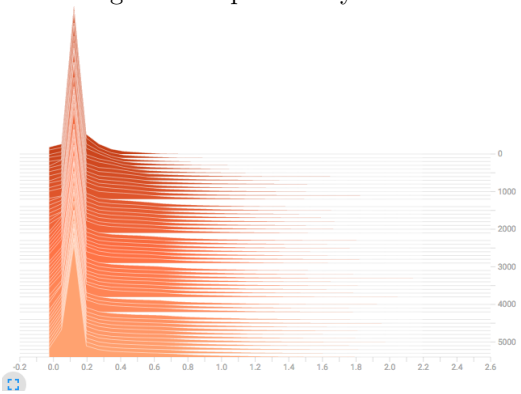

Figure 77: Histogram of activations after max pooling in layer one for all iterations

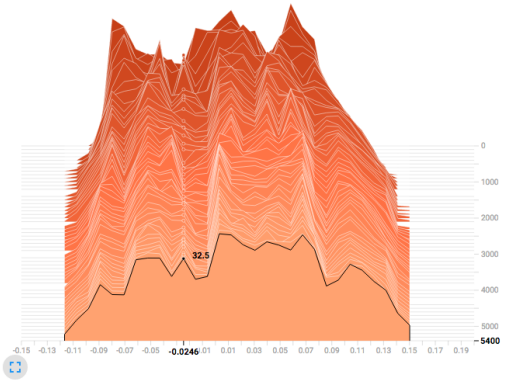

Figure 78: Histogram of weights in layer one for all iterations

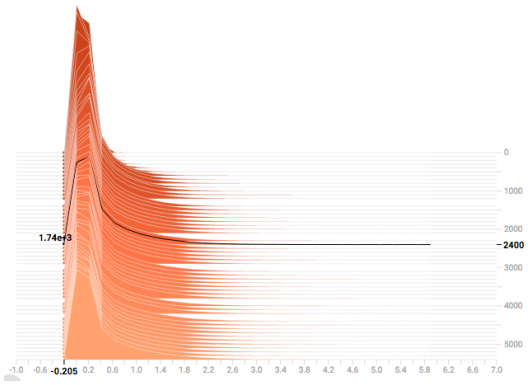

Figure 79: Histogram of activations in layer one for all iterations

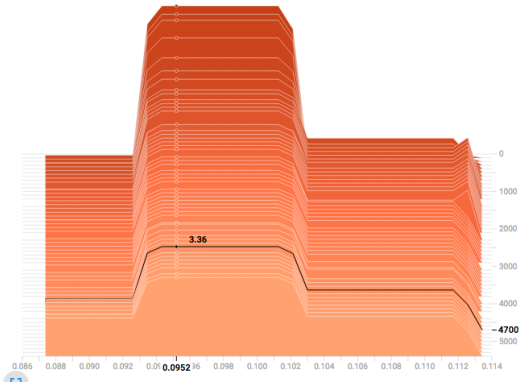

Figure 80: Histogram of biases in layer one for all iterations

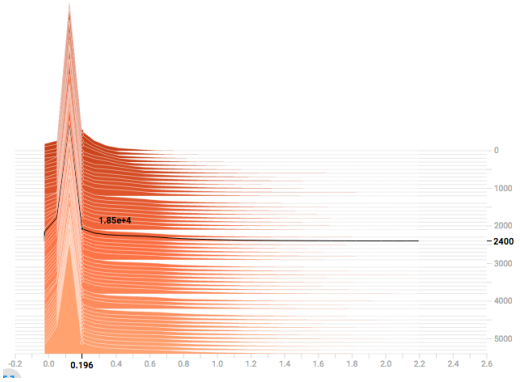

Figure 81: Histogram of inputs in layer one for all iterations

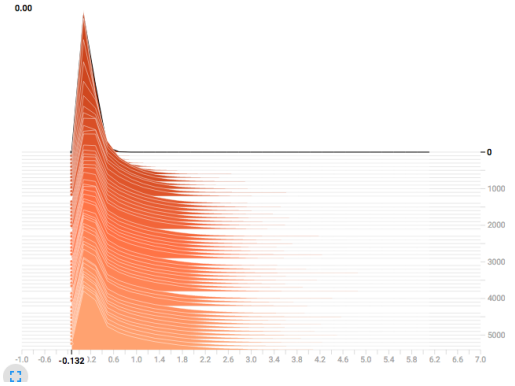

Figure 82: Histogram of activations after max pooling in layer one for all iterations

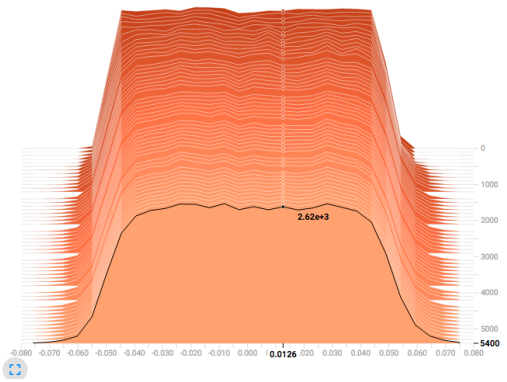

Figure 83: Histogram of weights in layer one for all iterations

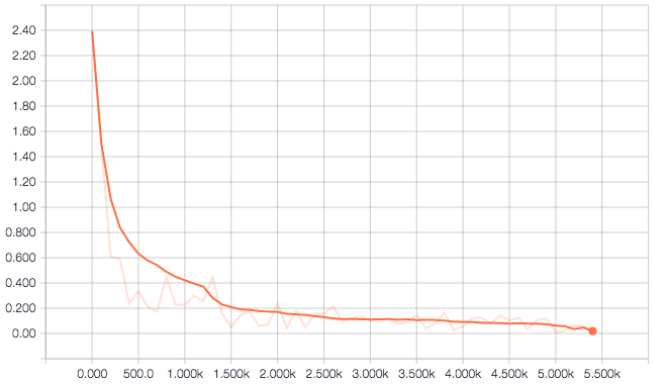

Figure 84: Training Error for all iterations

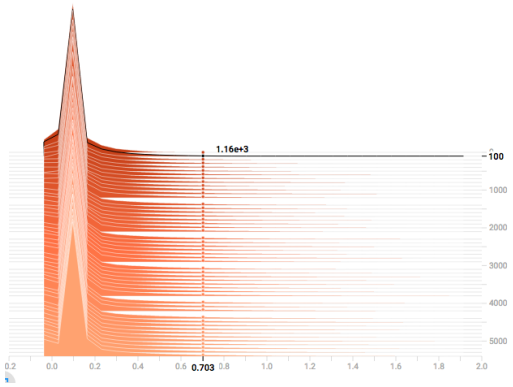

Figure 85: Histogram of activations in layer one for all iterations

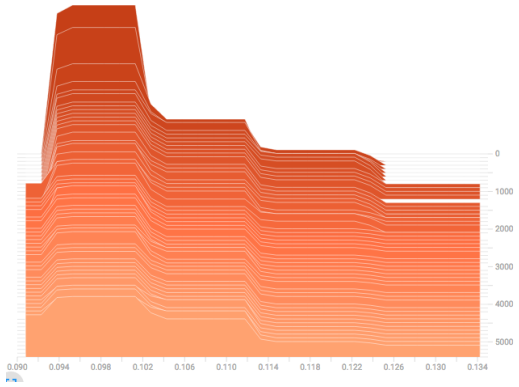

Figure 86: Histogram of biases in layer one for all iterations

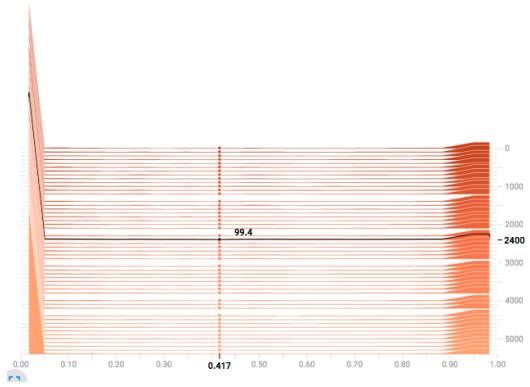

Figure 87: Histogram of inputs in layer one for all iterations

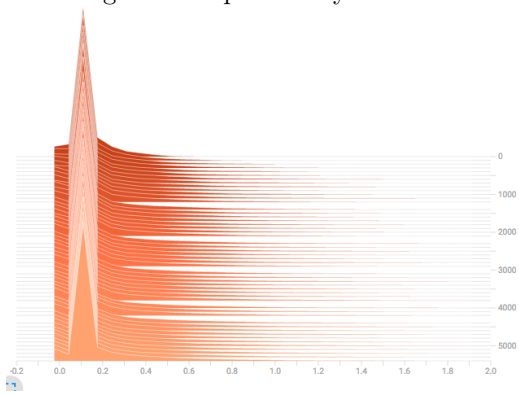

Figure 88: Histogram of activations after max pooling in layer one for all iterations

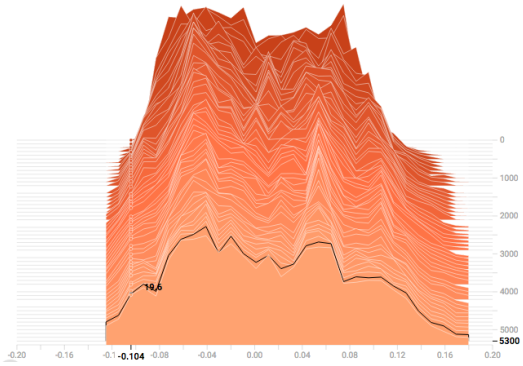

Figure 89: Histogram of weights in layer one for all iterations

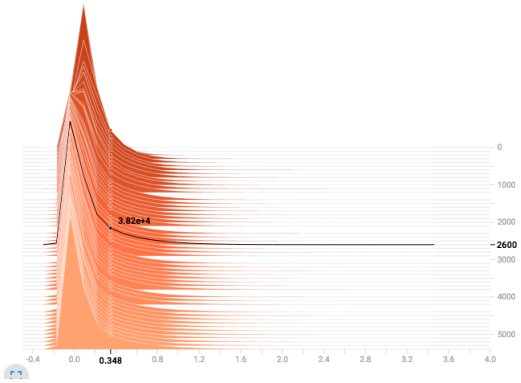

Figure 90: Histogram of activations in layer one for all iterations

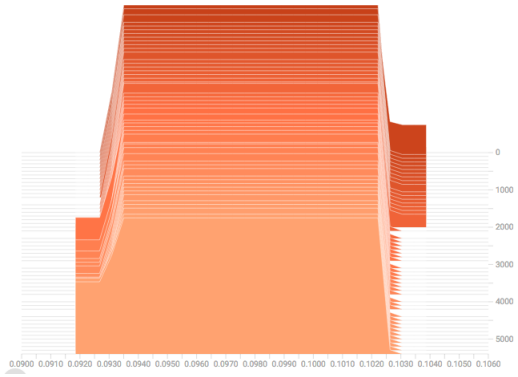

Figure 91: Histogram of biases in layer one for all iterations

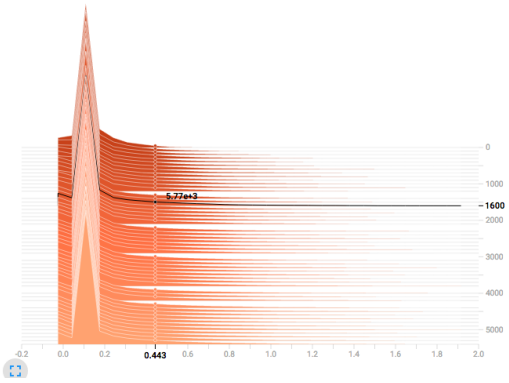

Figure 92: Histogram of inputs in layer one for all iterations

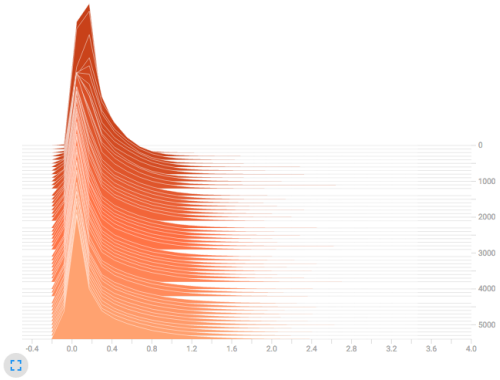

Figure 93: Histogram of activations after max pooling in layer one for all iterations

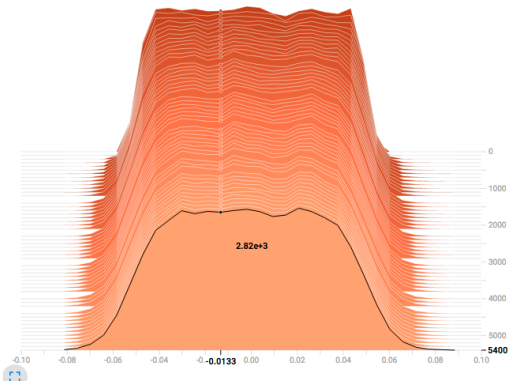

Figure 94: Histogram of weights in layer one for all iterations

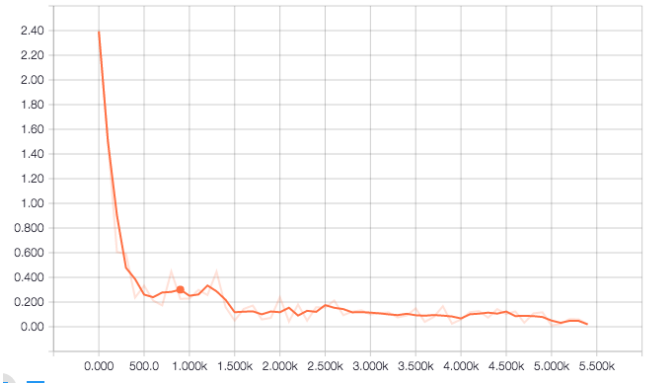

Figure 95: Training Error for all iterations