ELEC 576 / COMP 576: Fall 2021: Assignment 1

Youssaf Menacer

October 1, 2021

1 Backpropagation in a Simple Neural Network

1.1 Dataset

We will use the Make-Moons dataset available in Scikit-learn. Data points in this dataset form two interleaving half circles corresponding to two classes (e.g. female and male).

In the figure below, we generate and visualize Make-Moons dataset.

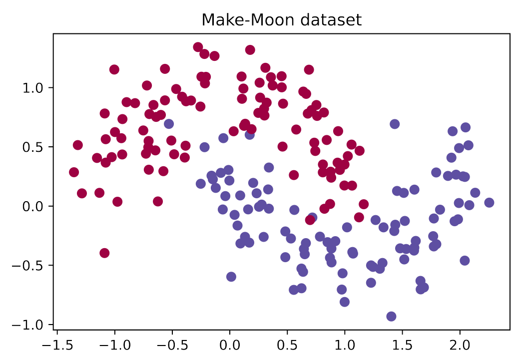


Figure 1: Make-Moons dataset

1.2 Activation Function

Here we implement Tanh, Sigmoid and ReLU activation functions and their derivatives used in neural networks.

In my code, I denote the activation functions by actFun(self, z, type). For type, we choose one of the three functions.

Similarly, for the derivative function diff-actFun(self, z, type).

1.3 Build the Neural Network

We now build a 3-layer neural network of one input layer, one hidden layer, and one output layer. The input to the network will be x- and y- coordinates and its output will be two probabilities, one for class 0 (female) and one for class 1 (male).

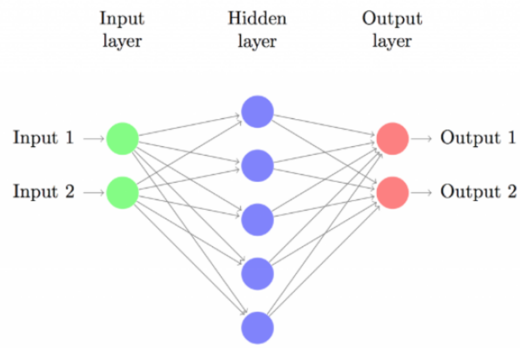


Figure 2: A 3-Layer NN

Here we have, 2-4-2 as the number of neurons (units) per layer.
In my code, the input vector is denoted by "X", and the target vector is denoted by "y".
We use "Softmax" as an activation function for the output layer, and we have

$$L(y, \hat{y}) = \sum_{n \in N} \sum_{i \in C} y_{n,i} log(\hat{y}_{n,i})$$

as our entropy loss function.
Where y are one-hot-encoding vectors and $\hat{y}$ are vectors of probabilities.

1.4 Backward Pass - Backpropagation

Here we implement the function backprop(self, X, y) after calculating the partial derivatives.

1.5 Time to Have Fun - Training!

Now we start training using boundary(self, X, y) function to visualize the decision boundary.

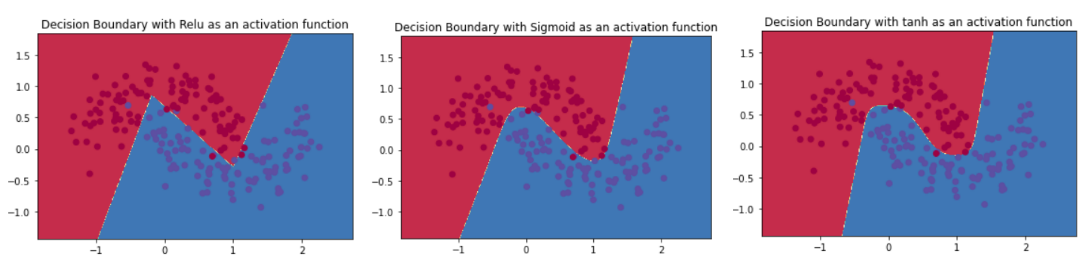


Figure 3: Decision Boundary with 'Relu','Sigmoid', and 'Tanh' as activation functions

Figure 3 shows the decision boundary with the three activation functions.

Now, in the next figure we show the convergence of the Loss function with each activation function.

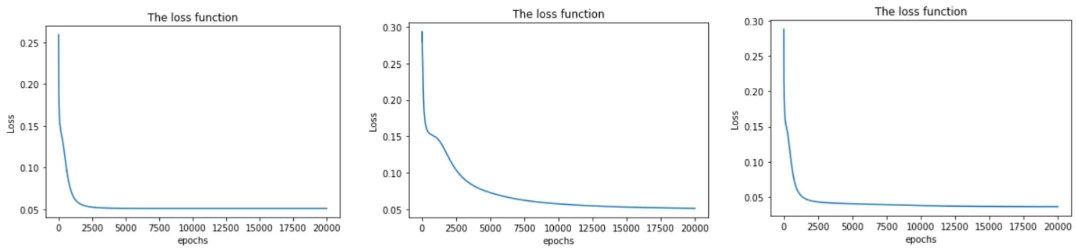


Figure 4: The Loss functions with 'Relu','Sigmoid', and 'Tanh' as activation functions

Figure 4 shows that the loss function decreases faster in two cases of "Relu" and "Tanh", but with "Sigmoid" we see a smooth decreasing. Moreover, the training ends faster in for "Relu" and "Tanh", but not for "Sigmoid".

Now, we increase the number of hidden layers, and we get the following results

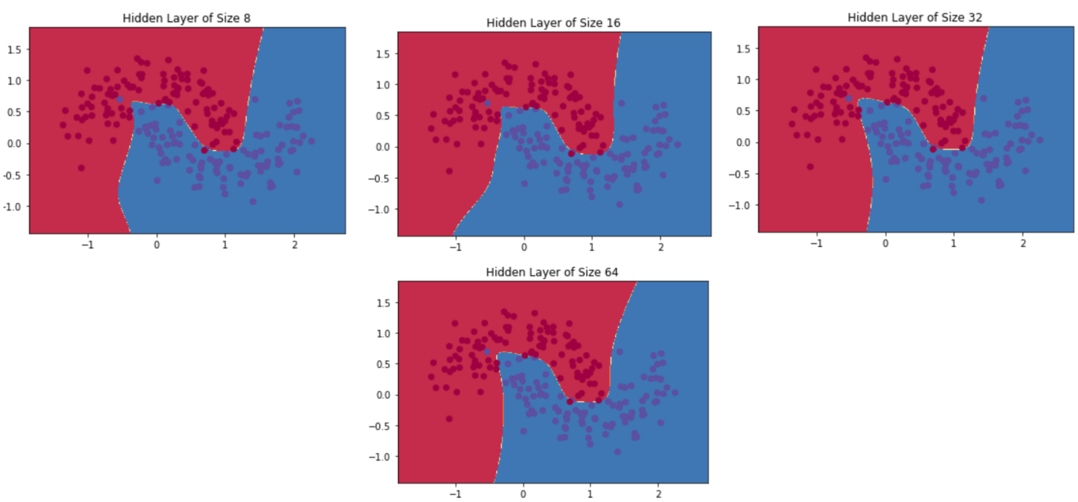


Figure 5: Decision Boundary with 8,16,32 and 64 hidden layers and "Tanh" activation function

We conclude from figure 5 that when the nmber of hidden layers get larger, we get an over-fitting of the data at the boundary. The main reason for this bug is the memorization of training data. Moreover, one can note that our NN model works better in training than testing data.

1.6 Even More Fun - Training a Deeper Network!!!

In the following figure I used a CPU in the first case for two hidden layers of 8-16 units for each. Moreover, I start with a lower learning rate, lr=0.06 and a number of iterations of 1000. Then, I used the GPU (Colab) for,

Case 01: Two hidden layers with 8-16 units in each layer with a lr=0.5 and nmuber of epochs= 10000.

Case 02: Two hidden layers with 16-32 units in each layer with a lr=0.8 and nmuber of epochs= 100000.

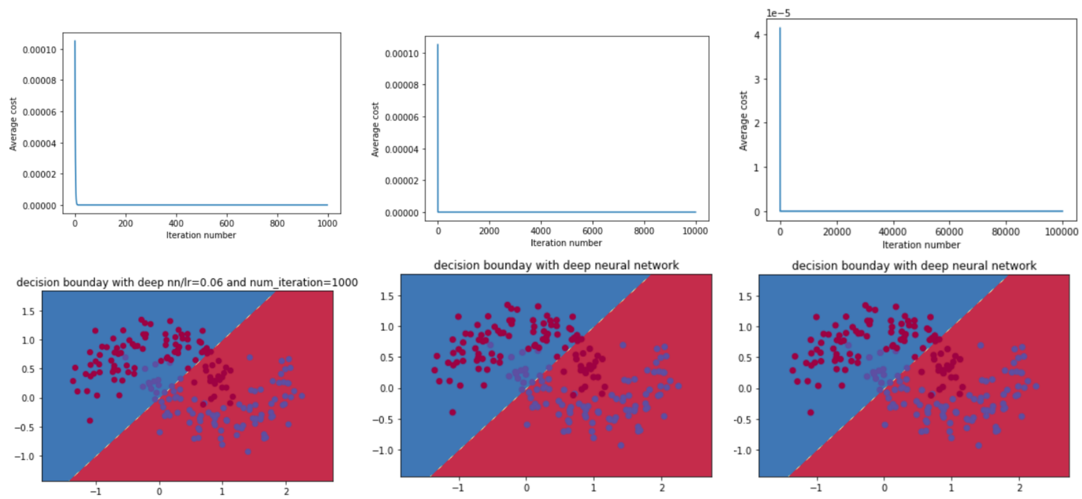


Figure 6: Decision Boundary with Deep Neural Network

I do not observe a big difference between the three cases. Also, the training ends up very fast with the slowest case (case01 with a CPU). The code is associated for this challenge.

2 Training a Simple Deep Convolutional Network on MNIST

We implement DCN following this architecture,

```
conv1(5-5-1-32) - ReLU - maxpool(2-2) - conv2(5-5-32-64) - ReLU - maxpool(2-2)
- fc(1024) - ReLU - DropOut(0.5) - Softmax(10)
```

2.1 Build and Train a 4-layer DCN

After running the code we got: test accuracy=0.9825 and the training takes 650.271767 second to finish.

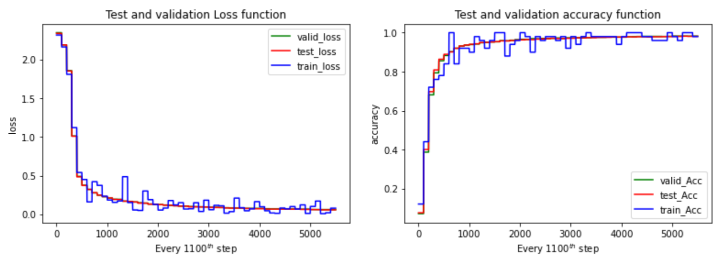
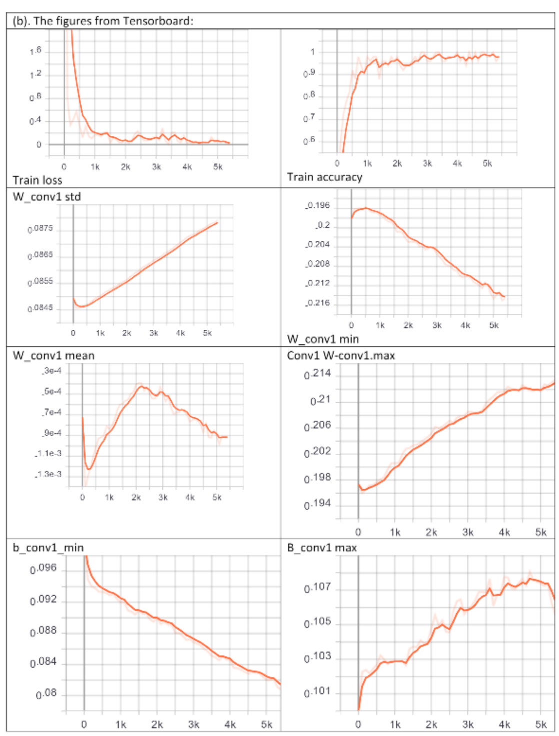


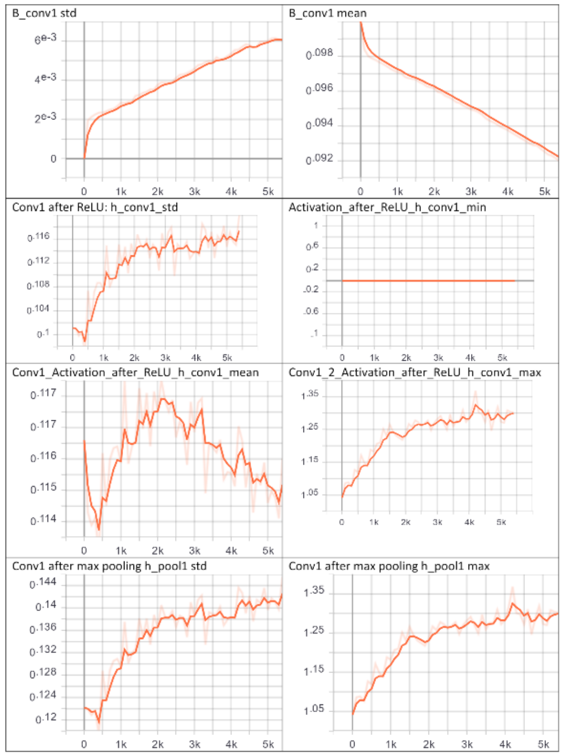
Figure 7: The loss and accuracy plots of training associated with validation and test

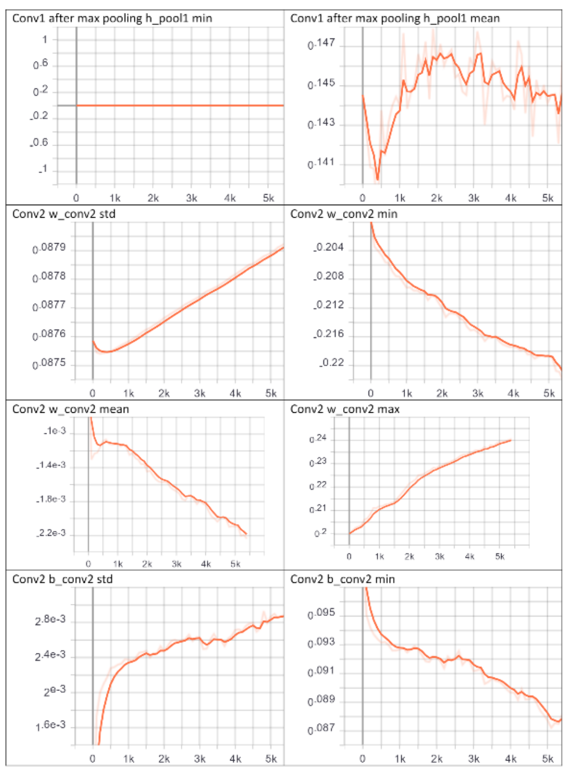
Figure 7 shows training loss function decreases while training accuracy increases. Moreover, validation and test loss follows the training loss. Similarly, for the test accuracy.

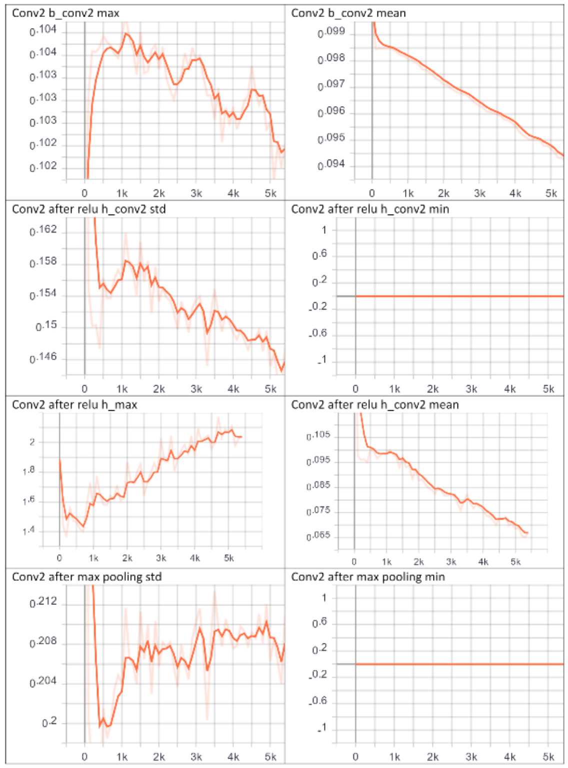
2.2 More on Visualizing Your Training

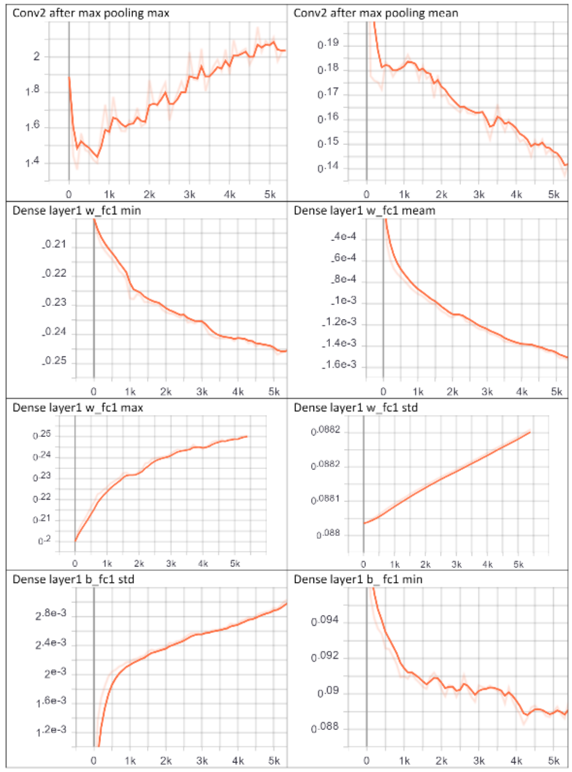
Here we modify dcn mnist.py so that one can monitor the statistics (min, max, mean, standard deviation, histogram) of the following terms after each 100 iterations: weights, biases, net inputs at each layer , activation after ReLU at each layer, activation after Max-Pooling at each layer.

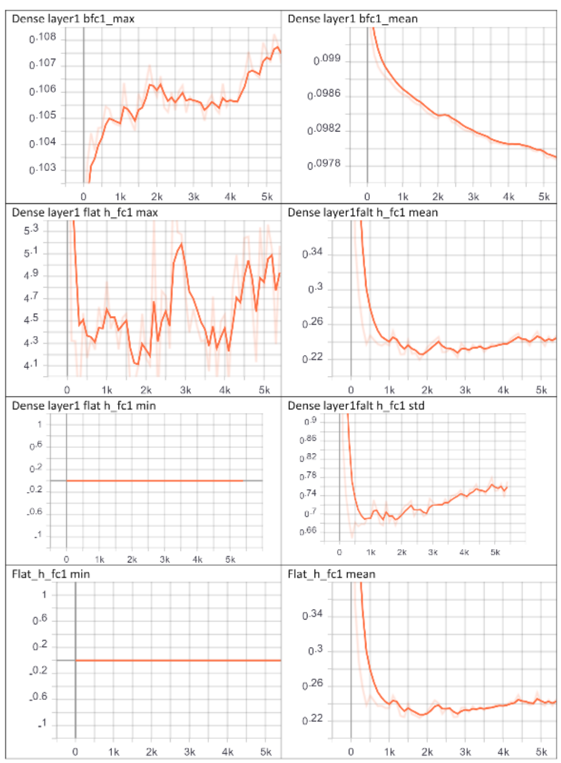


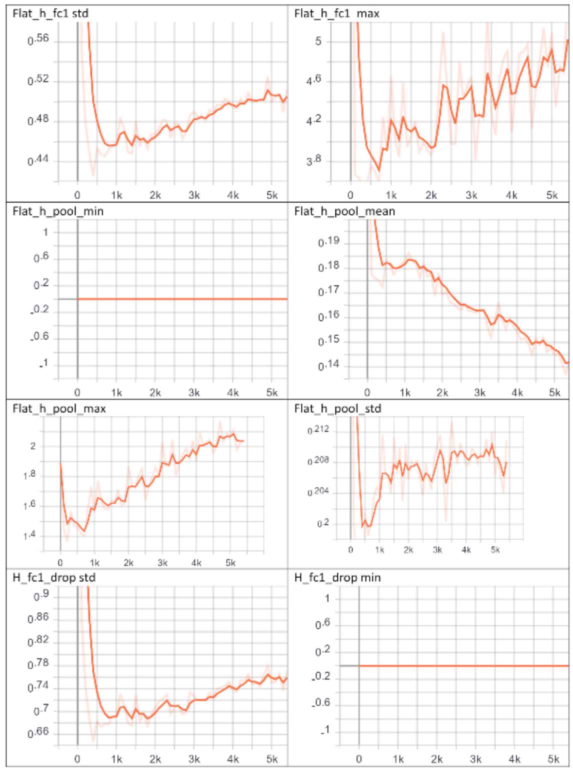

B_conv1 std
B_conv1 mean
Conv1 after ReLU: h_conv1_std
Activation_after_ReLU_h_conv1_min
Conv1_Activation_after_ReLU_h_conv1_mean
Conv1_2_Activation_after_ReLU_h_conv1_max
Conv1 after max pooling h_pool1 std
Conv1 after max pooling h_pool1 max

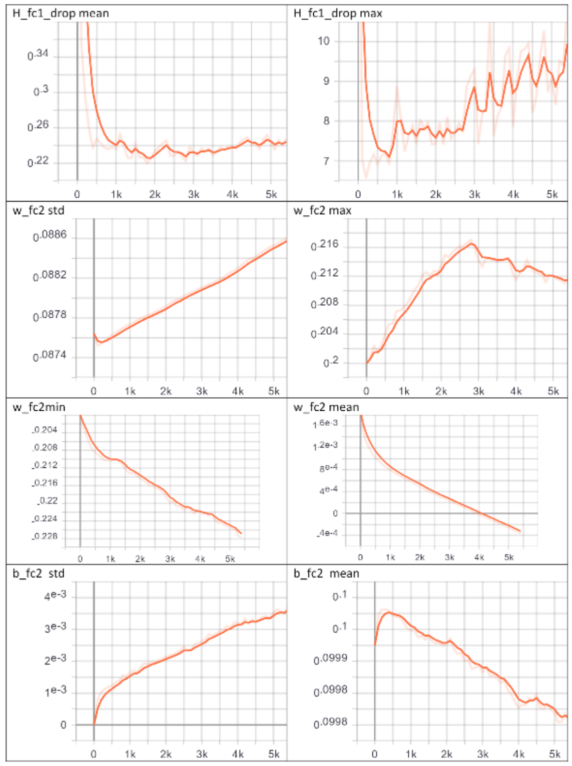

Conv1 after max pooling h_pool1 min
Conv1 after max pooling h_pool1 mean
Conv2 w_conv2 std
Conv2 w_conv2 min
Conv2 w_conv2 mean
Conv2 w_conv2 max
Conv2 b_conv2 std
Conv2 b_conv2 min

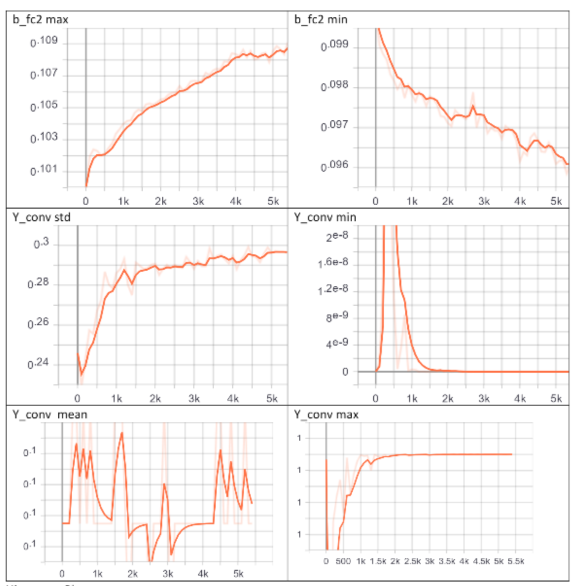

Conv2 b_conv2 max
Conv2 b_conv2 mean
Conv2 after relu h_conv2 std
Conv2 after relu h_conv2 min
Conv2 after relu h_max
Conv2 after relu h_conv2 mean
Conv2 after max pooling std
Conv2 after max pooling min


Conv2 after max pooling max
Conv2 after max pooling mean
Dense layer1 w_fc1 min
Dense layer1 w_fc1 meam
Dense layer1 w_fc1 max
Dense layer1 w_fc1 std
Dense layer1 b_fc1 std
Dense layer1 b_ fc1 min

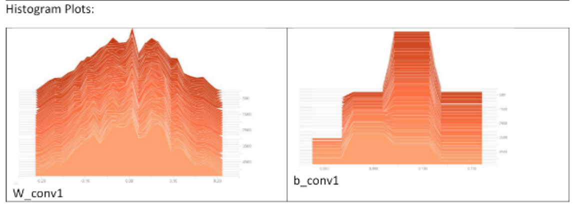

Dense layer1 bfc1_max
Dense layer1 bfc1_mean
Dense layer1 flat h_fc1 max
Dense layer1falt h_fc1 mean
Dense layer1 flat h_fc1 min
Dense layer1falt h_fc1 std
Flat_h_fc1 min
Flat_h_fc1 mean

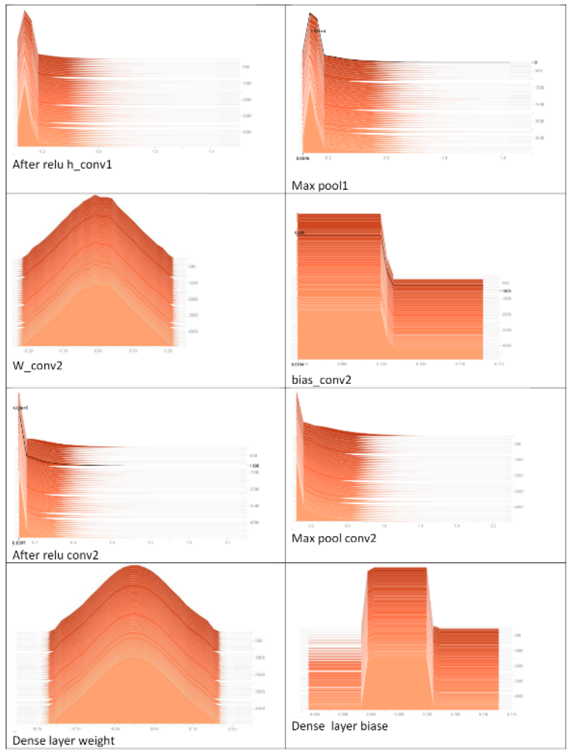

Flat_h_fc1 std
Flat_h_fc1 max
Flat_h_pool_min
Flat_h_pool_mean
Flat_h_pool_max
Flat_h_pool_std
H_fc1_drop std
H_fc1_drop min

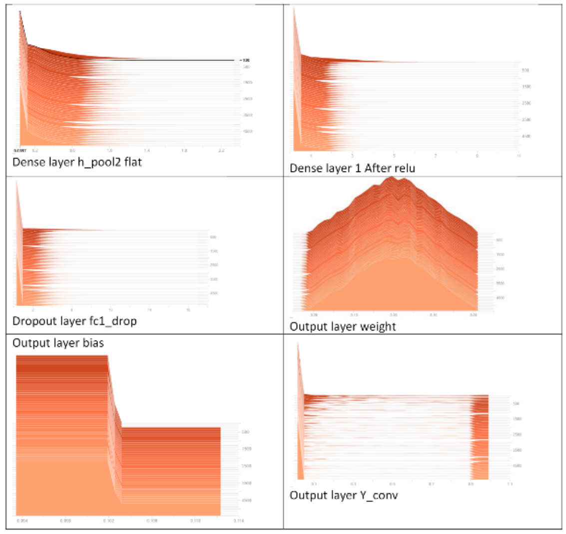




Histogram Plots:




After relu h_conv1
Max pool1
W_conv2
bias_conv2
After relu conv2
Max pool conv2
Dense layer weight
Dense layer biase



2.3 Time for More Fun!!!

In this section, we run the network training with different non- linearities (e.g: Leaky-ReLU), initialization techniques (Xavier) and training algorithms (e.g SGD).
The architecture for this part is as following,
conv1(5-5-1-32),leaky-Relu-maxpool(2,2)(kernel),
conv2(5,5,32,64),leaky-Relu, maxpool(2,2),fc(1024),leaky-Relu, DropOut(1/2),Softmax(10).
To initialize, I used Xavier technique and Monemtum optimizer to train the neural network.

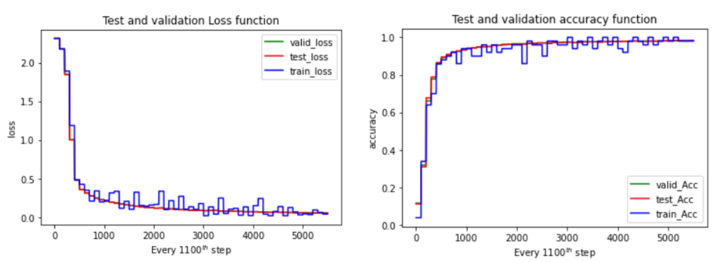
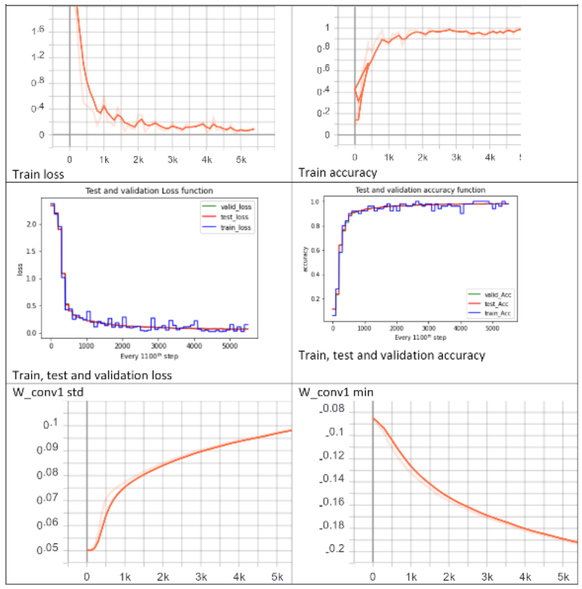
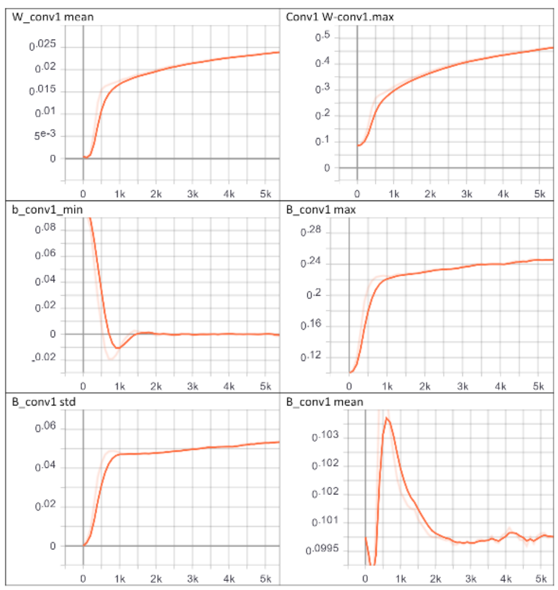
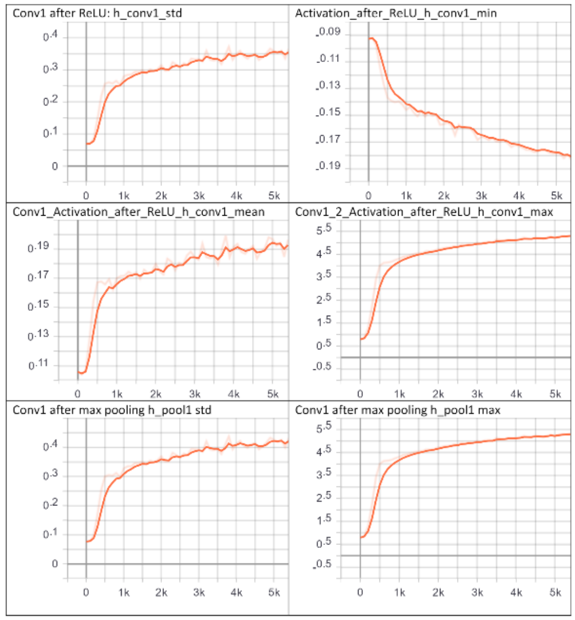


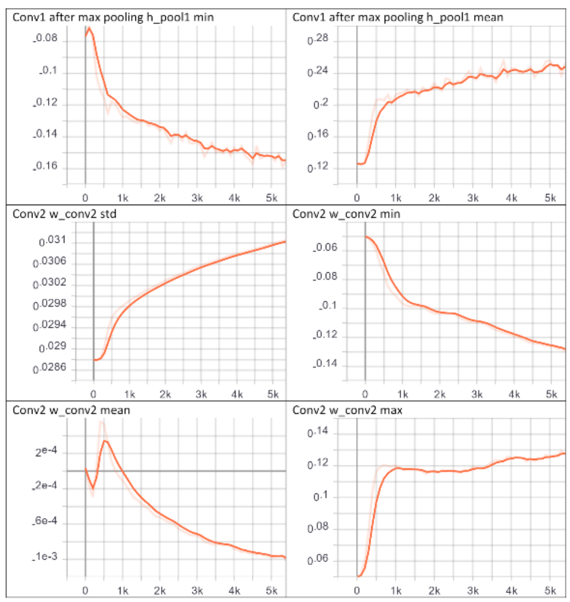
Figure 8: The loss and accuracy plots of training (validation/test)

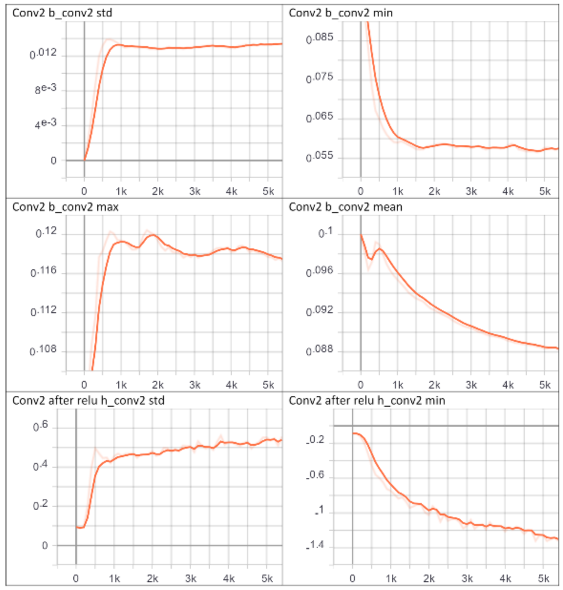
The test accuracy for this case is really good, 0.9822 in 676.062250 second to finish (CPU). I tried to used GPU(Google Colab, but there were some technical issues I was dealing with and never found a solution).

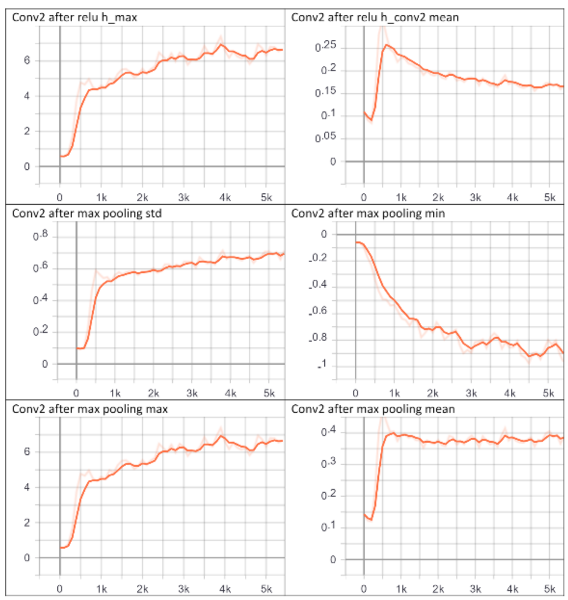

Train loss
Train accuracy
Test and validation Loss function
valid_loss
test_loss
train_loss
loss
Every 1100th step
Train, test and validation loss
Test and validation accuracy function
valid_Acc
test_Acc
train_Acc
accuracy
Every 1100th step
Train, test and validation accuracy
W_conv1 std
W_conv1 min

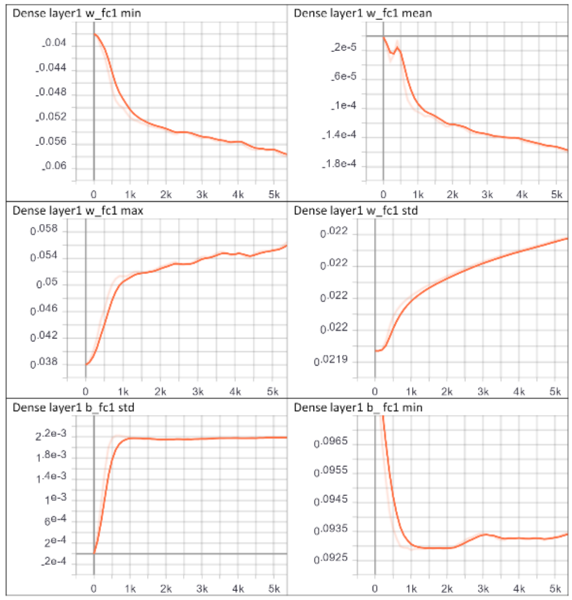

W_conv1 mean
Conv1 W-conv1.max
b_conv1_min
B_conv1 max
B_conv1 std
B_conv1 mean

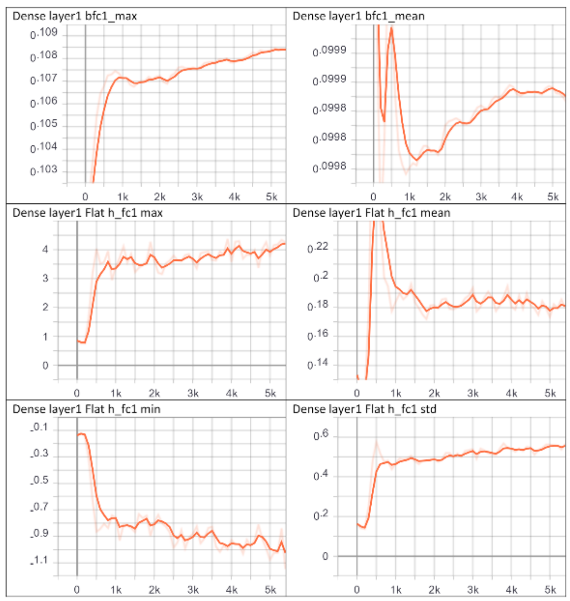

Conv1 after ReLU: h_conv1_std
Activation_after_ReLU_h_conv1_min
Conv1_Activation_after_ReLU_h_conv1_mean
Conv1_2_Activation_after_ReLU_h_conv1_max
Conv1 after max pooling h_pool1 std
Conv1 after max pooling h_pool1 max

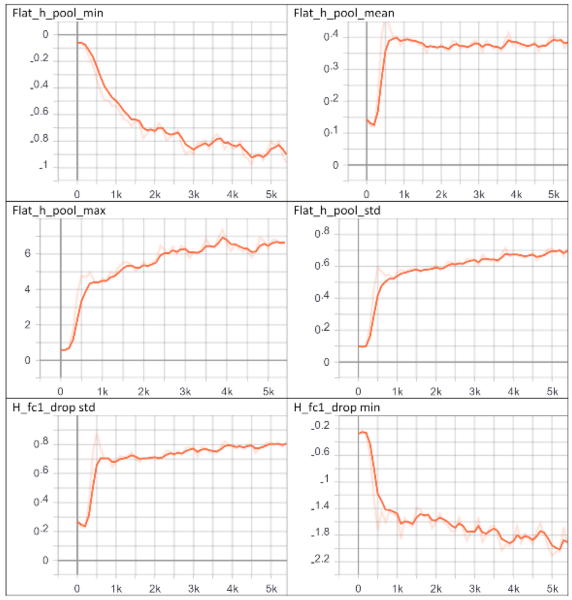

Conv1 after max pooling h_pool1 min
Conv1 after max pooling h_pool1 mean
Conv2 w_conv2 std
Conv2 w_conv2 min
Conv2 w_conv2 mean
Conv2 w_conv2 max

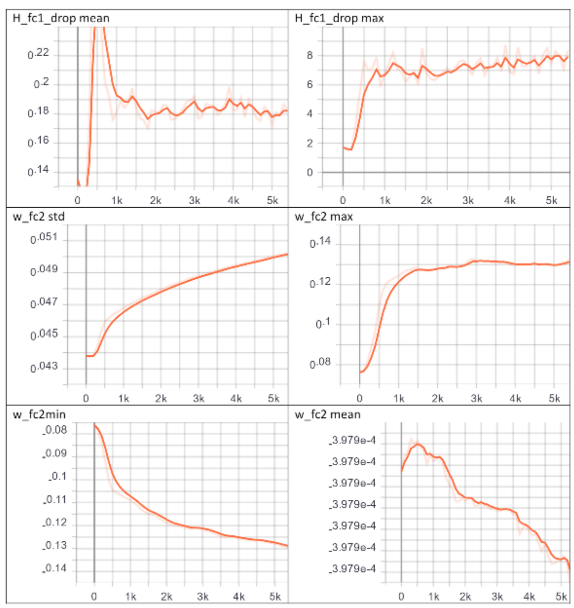

Conv2 b_conv2 std
Conv2 b_conv2 min
Conv2 b_conv2 max
Conv2 b_conv2 mean
Conv2 after relu h_conv2 std
Conv2 after relu h_conv2 min

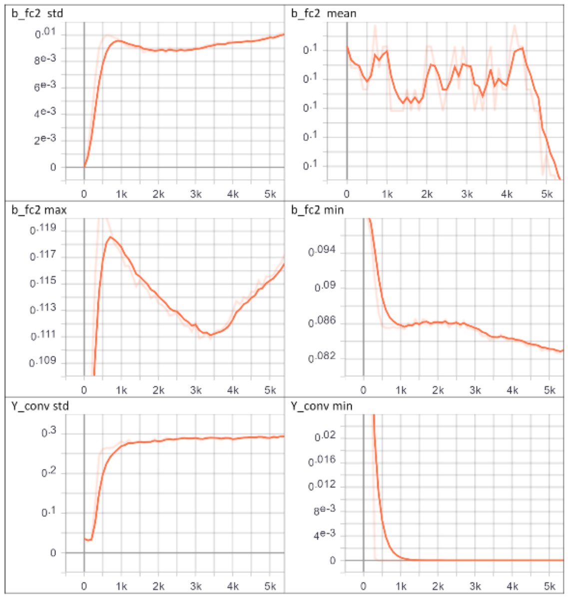

Conv2 after relu h_max
Conv2 after relu h_conv2 mean
Conv2 after max pooling std
Conv2 after max pooling min
Conv2 after max pooling max
Conv2 after max pooling mean

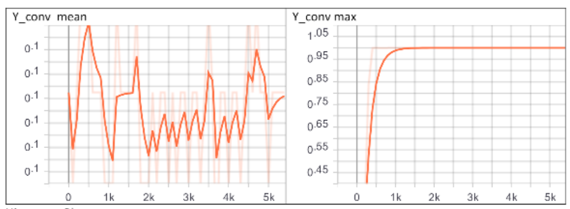

Dense layer1 w_fc1 min
-0.04
-0.044
-0.048
-0.052
-0.056
-0.06
Dense layer1 w_fc1 mean
-2e-5
-6e-5
-1e-4
-1.4e-4
-1.8e-4
Dense layer1 w_fc1 max
0.058
0.054
0.05
0.046
0.042
0.038
Dense layer1 w_fc1 std
0.022
0.022
0.022
0.022
0.0219
Dense layer1 b_fc1 std
2.2e-3
1.8e-3
1.4e-3
1e-3
6e-4
2e-4
-2e-4
Dense layer1 b_ fc1 min
0.0965
0.0955
0.0945
0.0935
0.0925
0
1k
2k
3k
4k
5k


Dense layer1 bfc1_max
0.109
0.108
0.107
0.106
0.105
0.104
0.103
Dense layer1 bfc1_mean
0.0999
0.0999
0.0998
0.0998
0.0998
Dense layer1 Flat h_fc1 max
4
3
2
1
0
Dense layer1 Flat h_fc1 mean
0.22
0.2
0.18
0.16
0.14
Dense layer1 Flat h_fc1 min
-0.1
-0.3
-0.5
-0.7
-0.9
-1.1
Dense layer1 Flat h_fc1 std
0.6
0.4
0.2
0
0
1k
2k
3k
4k
5k

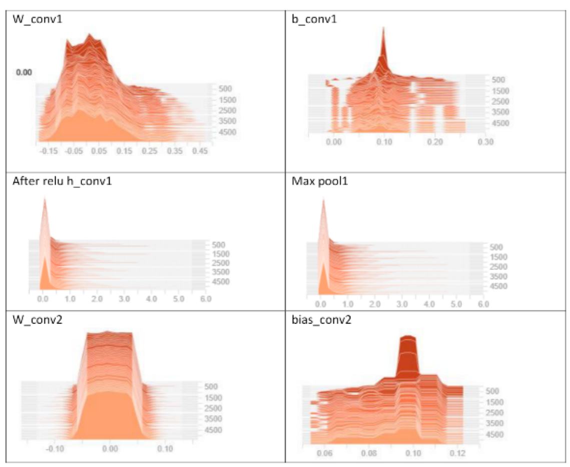

Flat_h_pool_min
Flat_h_pool_mean
Flat_h_pool_max
Flat_h_pool_std
H_fc1_drop std
H_fc1_drop min

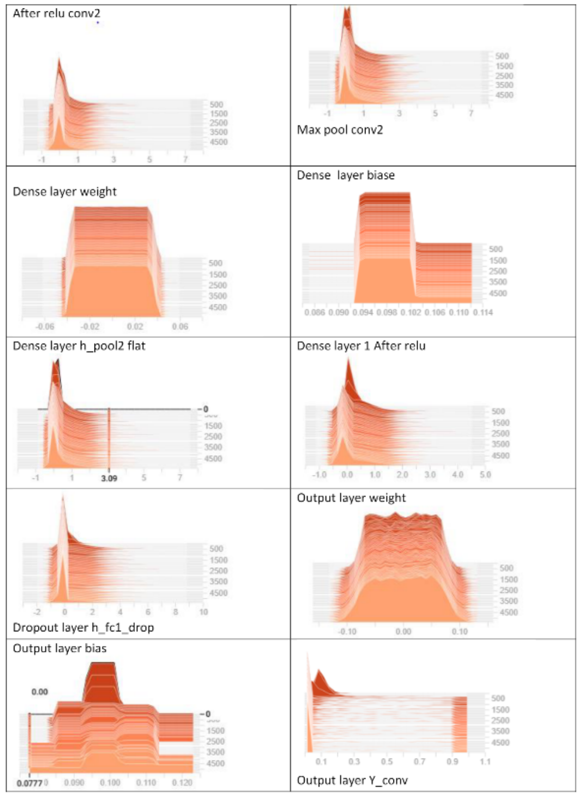

H_fc1_drop mean
H_fc1_drop max
w_fc2 std
w_fc2 max
w_fc2min
w_fc2 mean





Histogram Plots:





3 Resources

For this homework assignment, I used the following resources:
1/Introduction to Tensorflow: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture6.pdf
2/Introduction to DL/ML book: Pattern Recognition and Machine Learning's.Bishop
3/Back-propagation method based on lectures notes.
4/Neural Network/ CNN tutorials.