Name : Sait AKTURK

Student No : 21501734

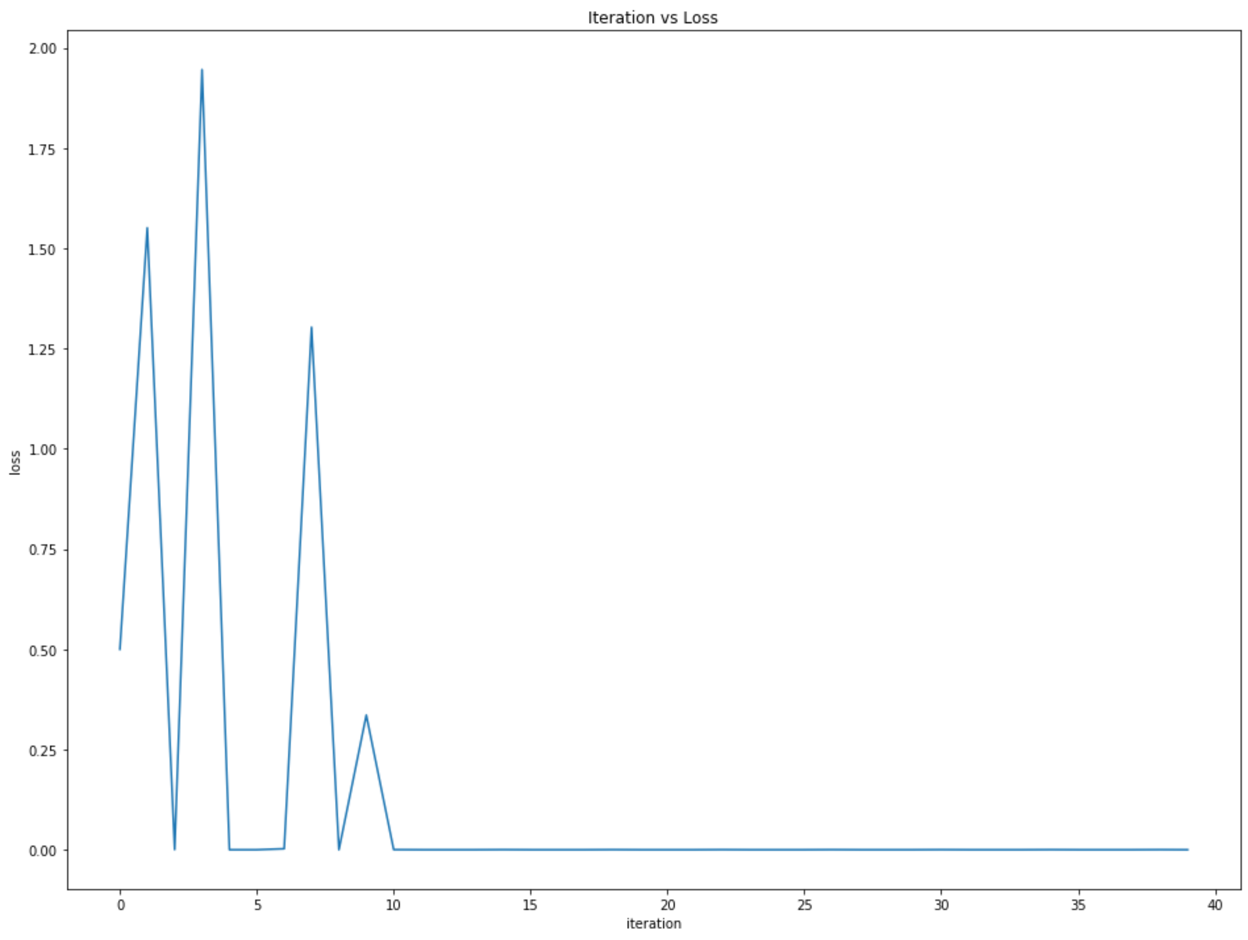
README

- **If you install matplotlib with pip, uninstall with ‘pip uninstall matplotlib’ and then install with conda ‘ conda install matplotlib’**
- I use scipy.io to read data, also use scipy.optimize for q4
- I use sklearn.metrics to get ‘confusion matrix’
- Lastly I use numpy library for general
- I suggest to tester to install all libraries with conda
- Put data in folder that code is inside of.

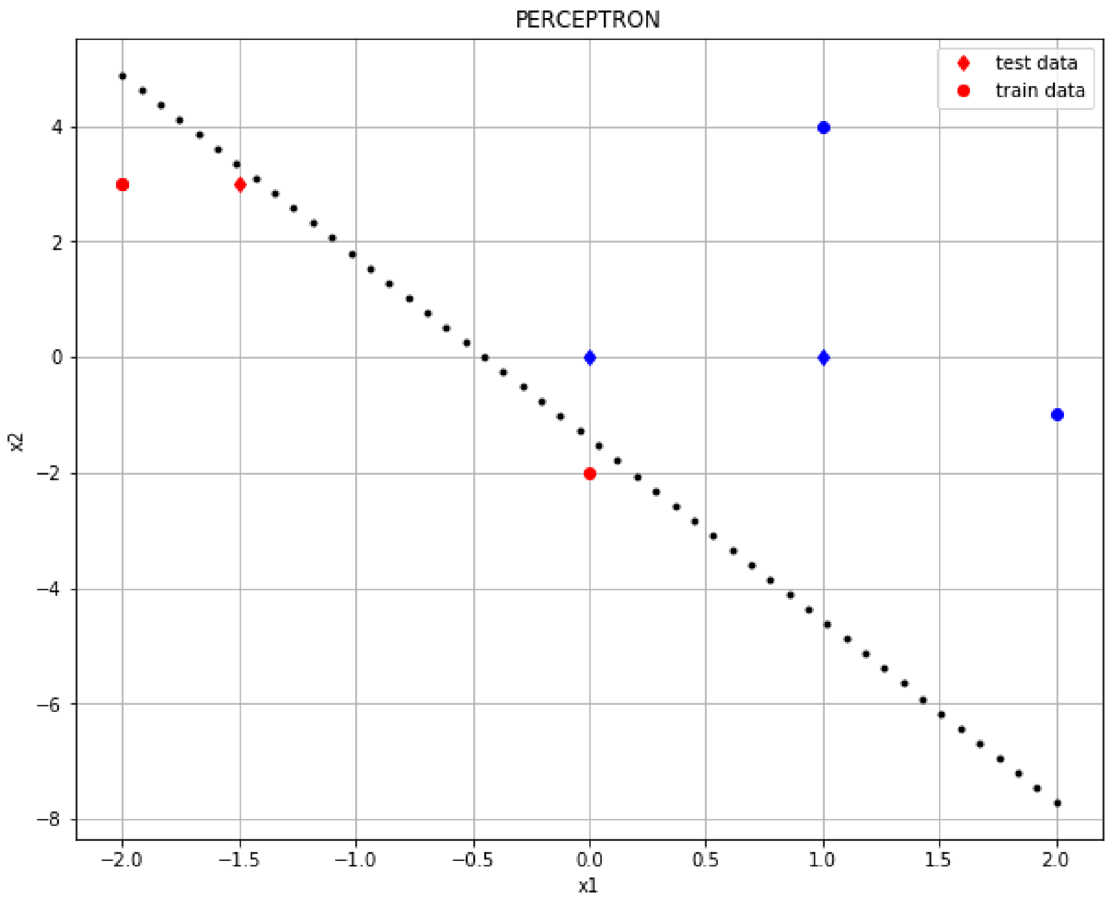
QUESTION 1

1.a) Since given data linearly separable, we could classify these data with only one decision boundary that one output unit will satisfy our expectations. Two input units are enough for two dimensional data. Since target bipolar and we want to choose activation function with bipolar and has-a-derivative property, I choose tanh() function.

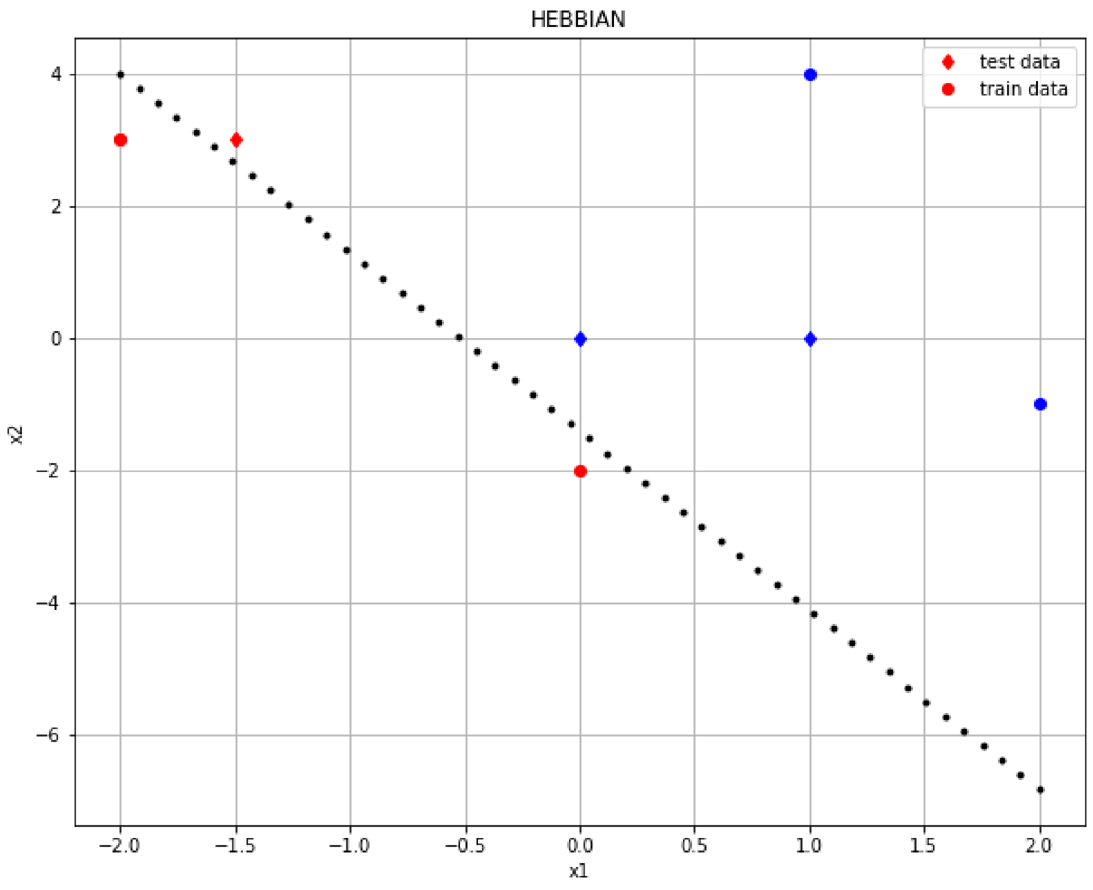
1.b) I am using online learning with single data point and I found that 3 iteration for each data point converge. Total 10 iteration could be said enough for approximation.



1.c) I think that decision boundary is not broad for detecting with good generalization test data '[-1.5,3]' classification is ambiguous because of decision boundary. (blue points represents '-1' and red points represent '1' ,thus, test data: [0,0] -> -1, [1,0] ->-1, [-1.5, 3] -> 1)


PERCEPTRON
test data
train data
x1
x2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
4
2
0
−2
−4
−6
−8

1.d) One iteration over entire train data was sufficient enough to classify data points. Only difference that test data point classified different than perception rule.



QUESTION 2

2.a)

Both cost function C_1 and C_2minimaziton required derivation that When we extend C_1:

I will use vectorized form of y_n and x_n that X is vectorized form x_n and Y is vectorized form of y_n

$$C_1 = \frac{1}{2}(Y - w^T X)(Y - w^T X)^T$$
$$= \frac{1}{2}(YY^T - YX^T w - w^T XY^T - w^T XX^T w)$$

We know that result of $w^T XY^T$ and $YX^T w$ is scalar and equal. Then we could write C_1

$$= \frac{1}{2}(Y^T Y - 2YX^T w + w^T XX^T w)$$

1

$$\underset{w}{argmin}\ C_1 \Rightarrow \frac{dC_1}{dw} = 0$$

Therefore we could ignore constant YY^T that independent from w

C_1 becomes :

$$\underset{w}{argmin}\ C_1 = \frac{1}{2} w^T XX^T w - YX^T w$$

From that we could observe that :

$$\underset{w}{argmin}\ C_1 = \frac{1}{2} w^T XX^T w - YX^T w$$

$$\underset{w}{argmin}\ C_2 = \frac{1}{2} w^T A w - b^T w$$

$$\boxed{\Rightarrow A = XX^T and\ \mathrm{b} = XY^T}$$

$$\boxed{\begin{aligned} \Delta w &= \frac{dC_1}{dw} = \frac{1}{2}(XX^T + X^T X)w - XY^T \\ \Delta w &= \frac{dC_2}{dw} = \frac{1}{2}(A + A^T)w - b \end{aligned}}$$

3.b)

Since we know that $\Delta\tilde{w}$ and Δw is equal.

$$\Delta\tilde{w} = \Delta w$$
$$\frac{1}{2}(A + A^T)\tilde{w} = \frac{1}{2}(A + A^T)w - b$$
$$2 * \frac{1}{2}(A + A^T)^{-1}(A + A^T)\tilde{w} = 2 * (A + A^T)^{-1}(\frac{1}{2}(A + A^T)w - b)$$
$$\tilde{w} = w - 2 * (A + A^T)^{-1} b$$

Proof :
From changes above, we can write C_3 as:

$$C_3 = 0.5 * (w - 2 * (A + A^T)^{-1}b)^T A(w - 2 * (A + A^T)^{-1}b)$$

$$\Rightarrow \frac{dC_3}{dw} = 0.5 * (A(w - 2 * (A + A^T)^{-1}b) + A^T(w - 2 * (A + A^T)^{-1}b))$$

$$\Rightarrow \frac{dC_3}{dw} = 0.5 * ((A + A^T)(w - 2 * (A + A^T)^{-1}b))$$

$$\Rightarrow \frac{dC_3}{dw} = (\frac{1}{2}(A + A^T)w - b)$$

From 3.a we can observe that we got same update rule with $\tilde{w}$

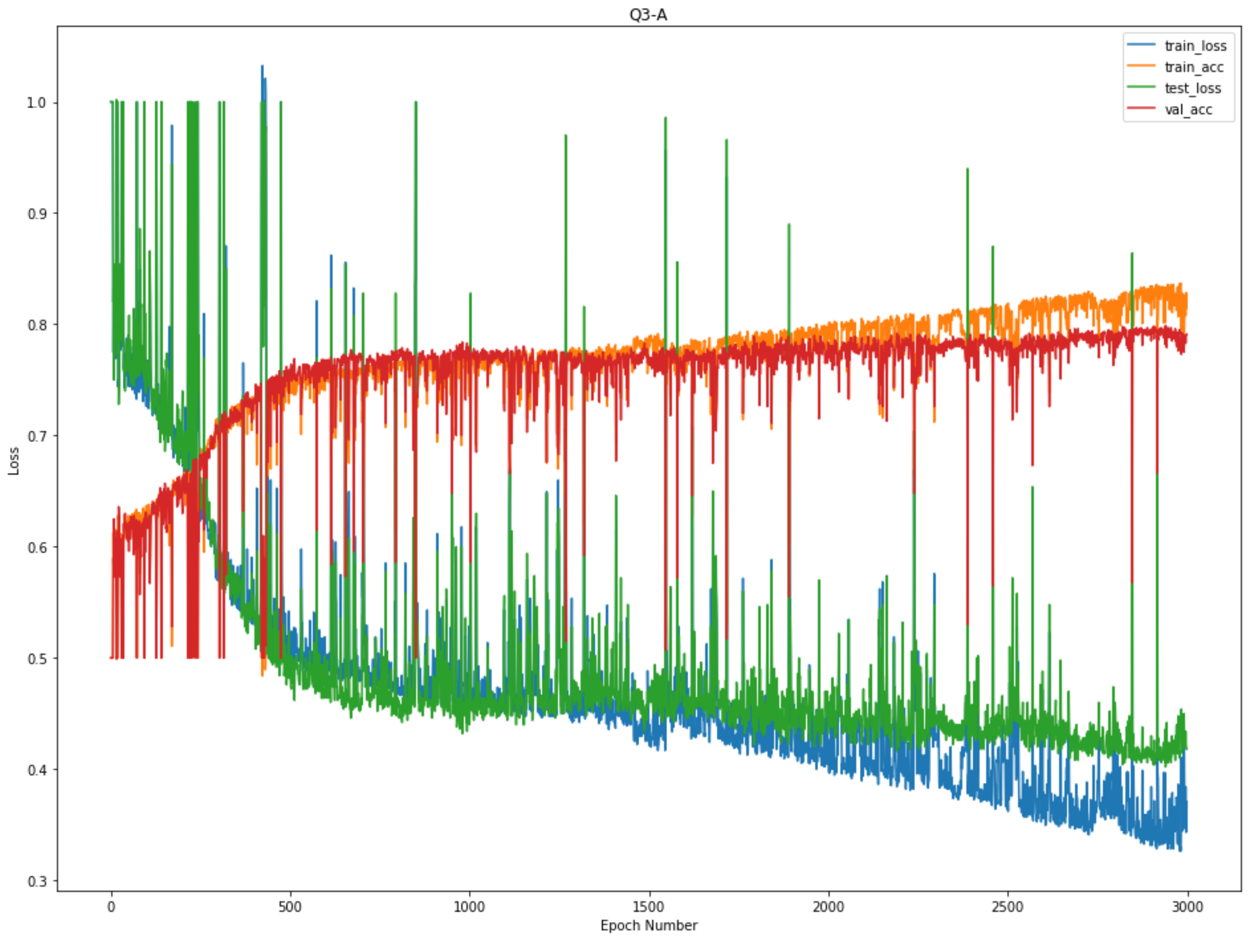
3.c) I believe this question is related to conjugate gradient method that when A is symmetric update rule becomes Aw=b that essentially linear system solutions. When we implement conjugate gradient descent, what we find is optimal step size, found with derivative of step size according to gradients. One of solution is Polak-Ribiere approach to find optimal step size.

$$\beta_n^{PR} = \frac{\Delta x_n^T(\Delta x_n - \Delta x_{n-1})}{\Delta x_{n-1}^T \Delta x_{n-1}}.$$

Where delta x_n is gradient for n of x value means weights, delta x_{n-1} is gradient for n-1 of x value means.

QUESTION 3

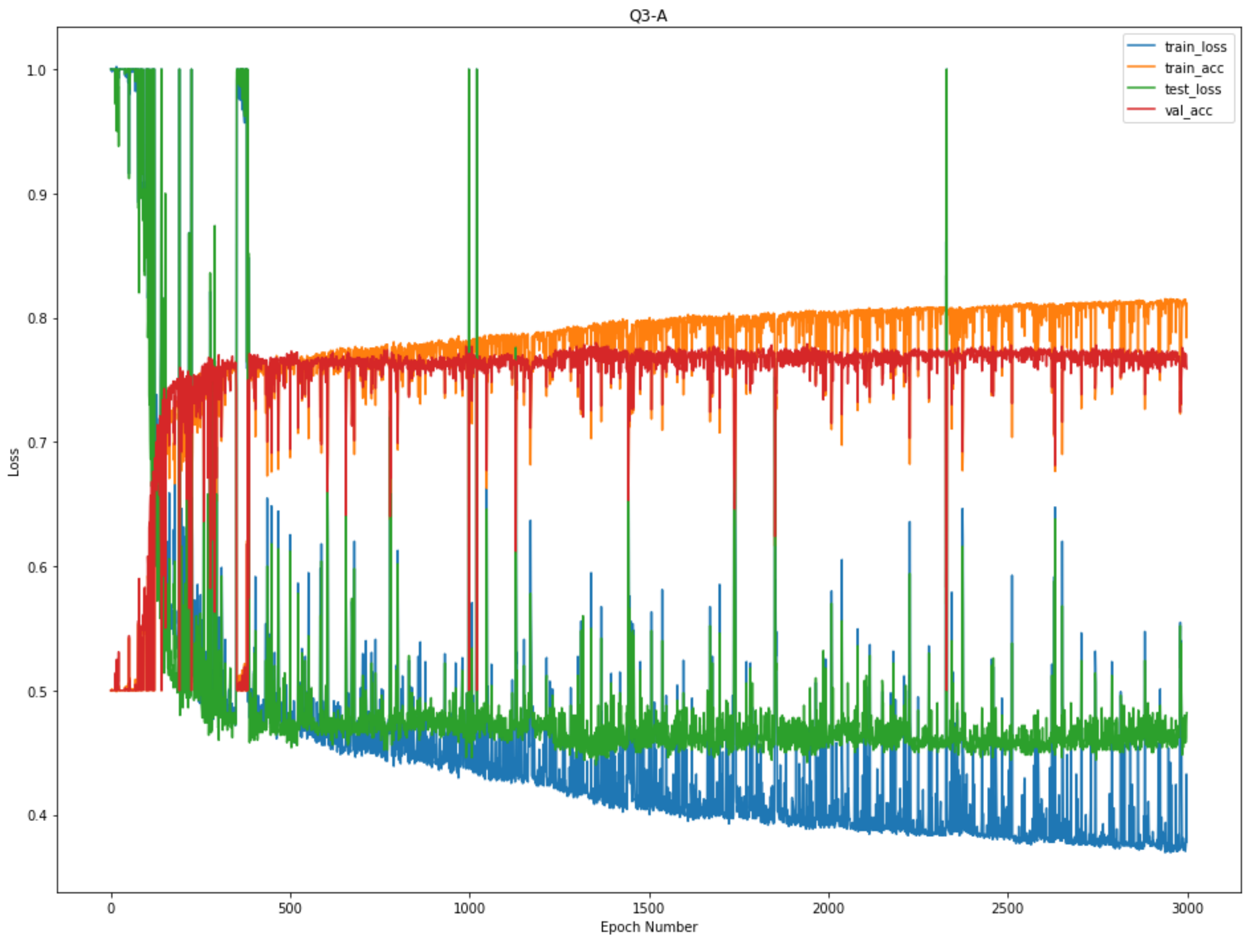
3.a) I use Xavier initialization that getting random number divide to square root of 1/layer$_{l-1}$ that subscript ‘l’ represent l-1 layer of network because I searched over all weight initialization that work well, I found that particularly Xavier initialization works well with tanh function. I also select learning rate as “0.4” to converge faster. Also, I select hidden unit size as “4” because little size for hidden layer helps to converge faster, for bigger size of hidden unit, the network slows down that never reach optimal point. I also select batch size as “24” to prevent overfitting on early epochs. I choose epoch number as “3000” that I saw that this epoch number was optimal epoch number because the network reach its peak points and after that iteration the network became overfit. As a result, I got %79-81 accuracy on test set.



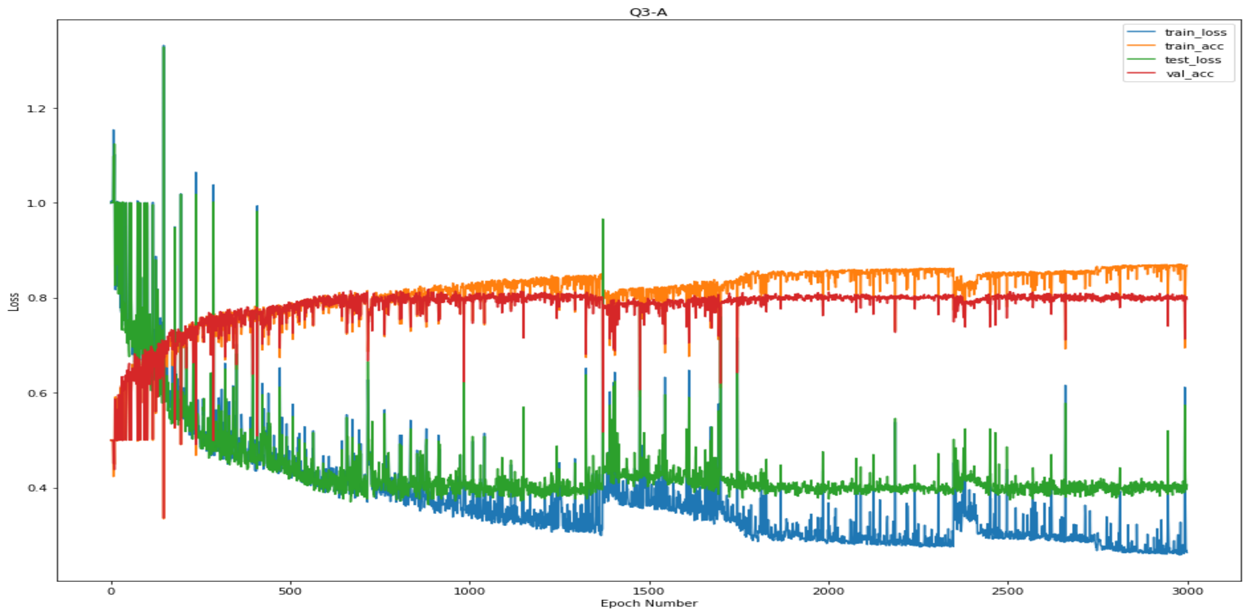
3.b) In first 500 epochs of training, both classification and squared error became converge some optimal points for both train and test data. After that, both train and test accuracy linearly increased and their loss linearly decreased. However, since I select shallow network to converge fast, the network became overfitting, thus I stop the iteration and noted that iteration number. As an answer for second part of question, I experimented and noticed that squared error could be representative of classification error because as you can see their converge behavior and peak time pair off each other, however, I could say that we could not know real accuracy with only looking squared error because different architecture and different data may give same squared error, however, could give different classification error.

3.c)

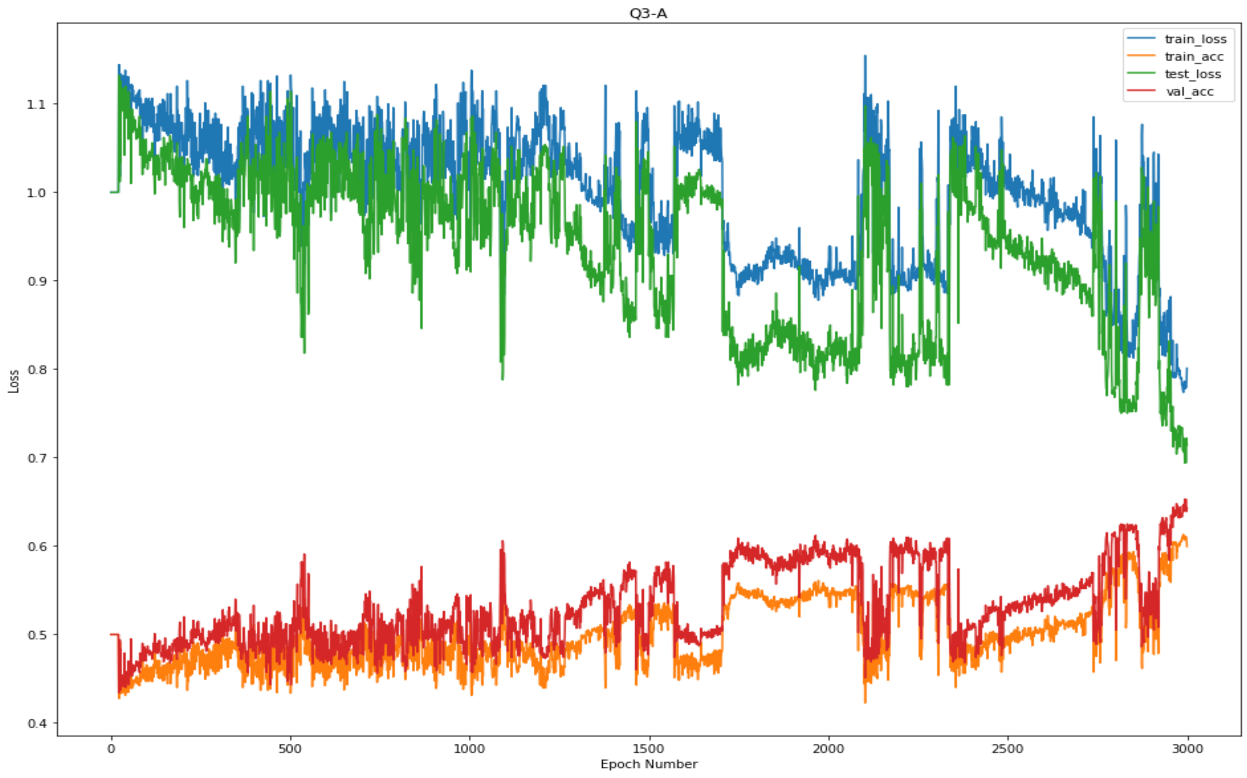
One hidden unit : Low



Four hidden unit : Optimal



64 hidden unit : High



3.d)

Learning rate : 0.25

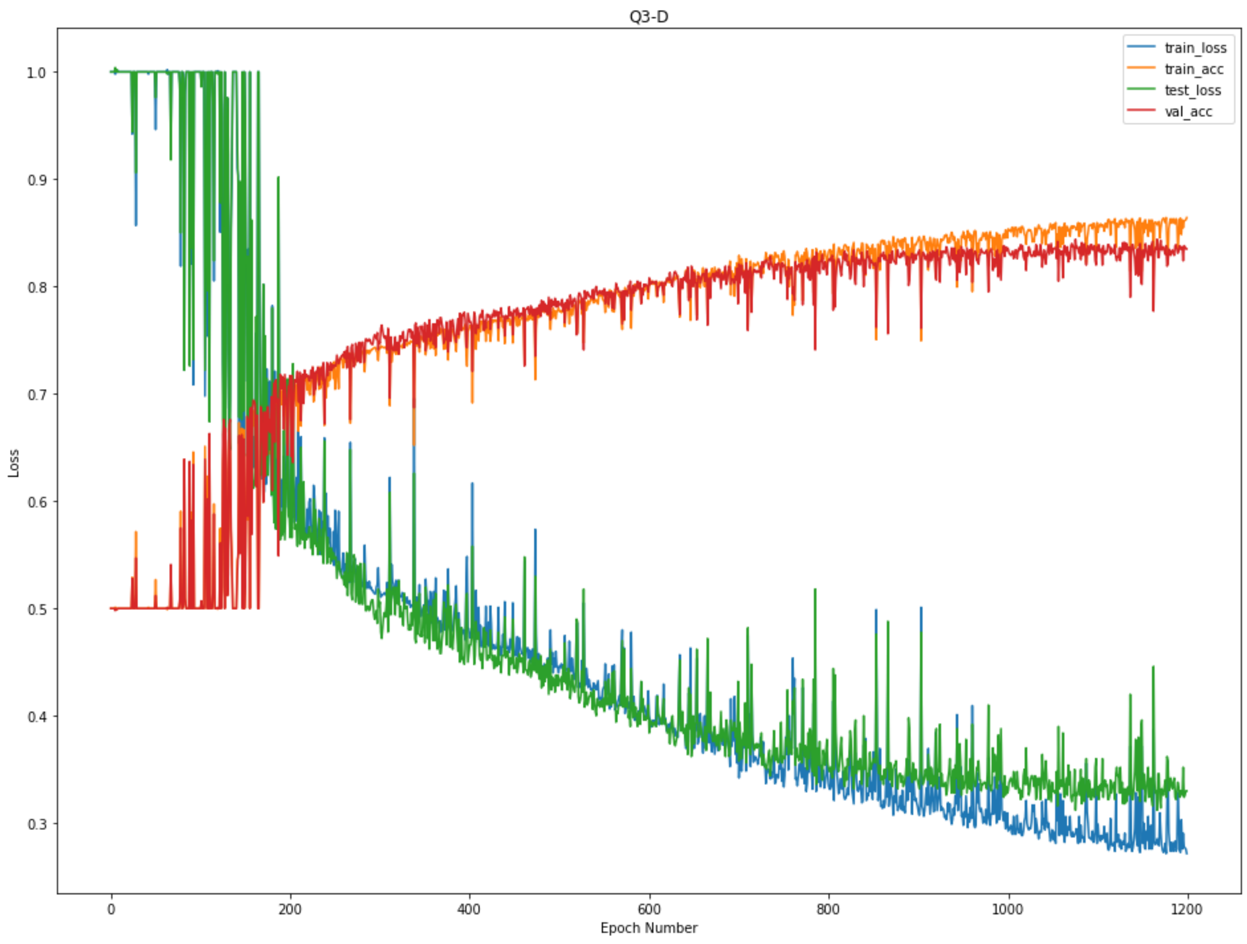
First Hidden Unit Size : 32

Second Hidden Unit Size : 8

Batch Size : 32

Epoch number : 1200

Two hidden layer help to increase accuracy over 3% that test accuracy that we obtain reach 82%-84%. This network reach optimal accuracy error faster than 3.a . Therefore, epoch number substantially lower than 3.a .



3.e)

Momentum rate : 0.5

Learning rate : 0.25

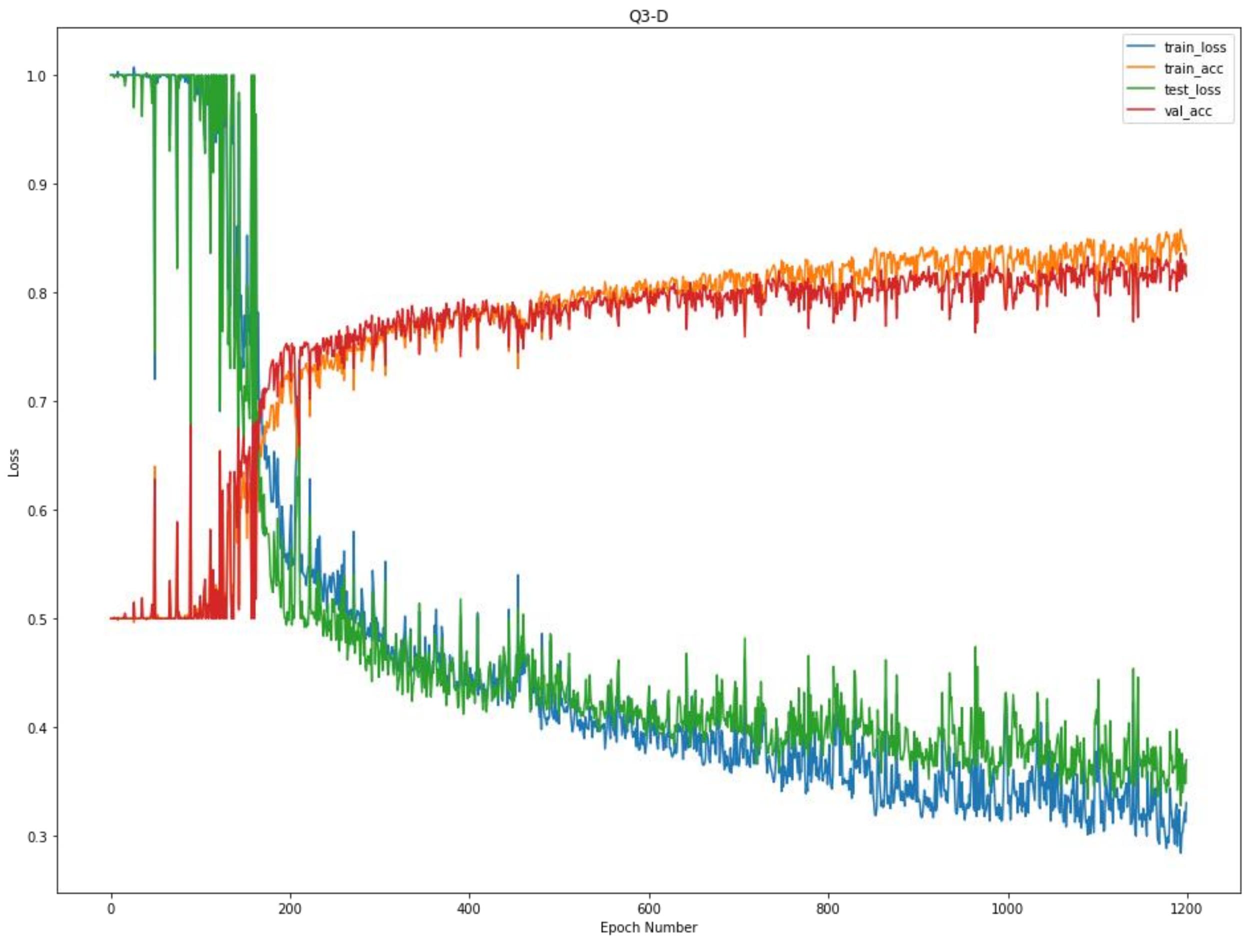
First Hidden Unit Size : 32

Second Hidden Unit Size : 8

Batch Size : 32

Epoch number : 1200

Comparatively, learning with momentum shows more stable converge towards to optimal error points. And also, momentum reduce the overfitting speed of the network.

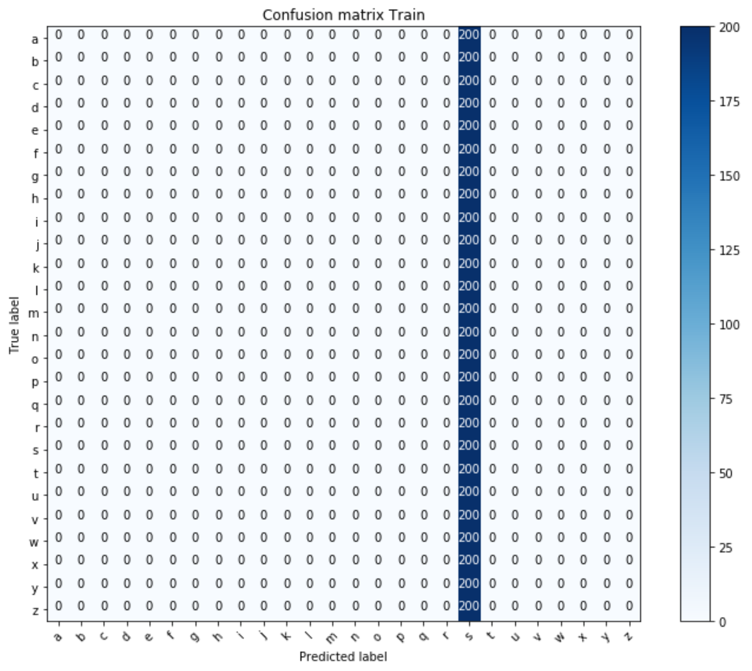


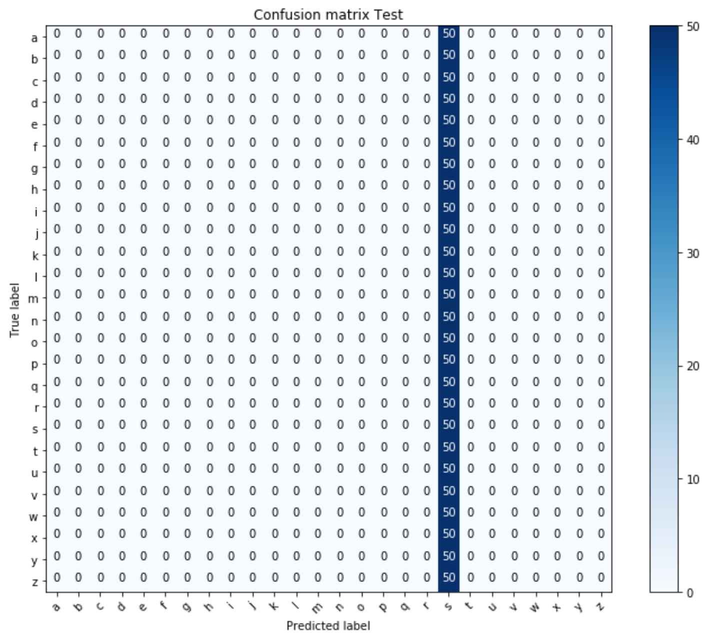
QUESTION 4

4.a)

Because I train my model with minibatch size = 1, the network heavily overfit.(you could run myscript to see separate images in your file). Because I don't know to use conjugate gradient descent in python 'scipy.optimize'. You can check in code that I wrote everythin, however, I could not find any solution. However, you can check implementation, I used multiclass cross entropy with softmax.

4.b)





Like I mentioned about overfitting, the neural network become biased or learn nothing useful so that could not classify the datas.

4.c) Because of my implementation is not working completely. Results are meaningless(but you can see next page).However, l2 regularization tries to reduce model weights towards 0 to prevent overfitting with training data, as I know, lambda values generally selected range between 0 and 1, however due to, conjugate overfit speed, we want to penalize the weights heavily so that the network never overfit. However, if my implementation work perfectly, I should expect more smooth images due to regularization and in the confusion matrix, I should also expect some underfitting or optimal results. The bigger the lambda, what I mentioned above , became bigger and obvious.

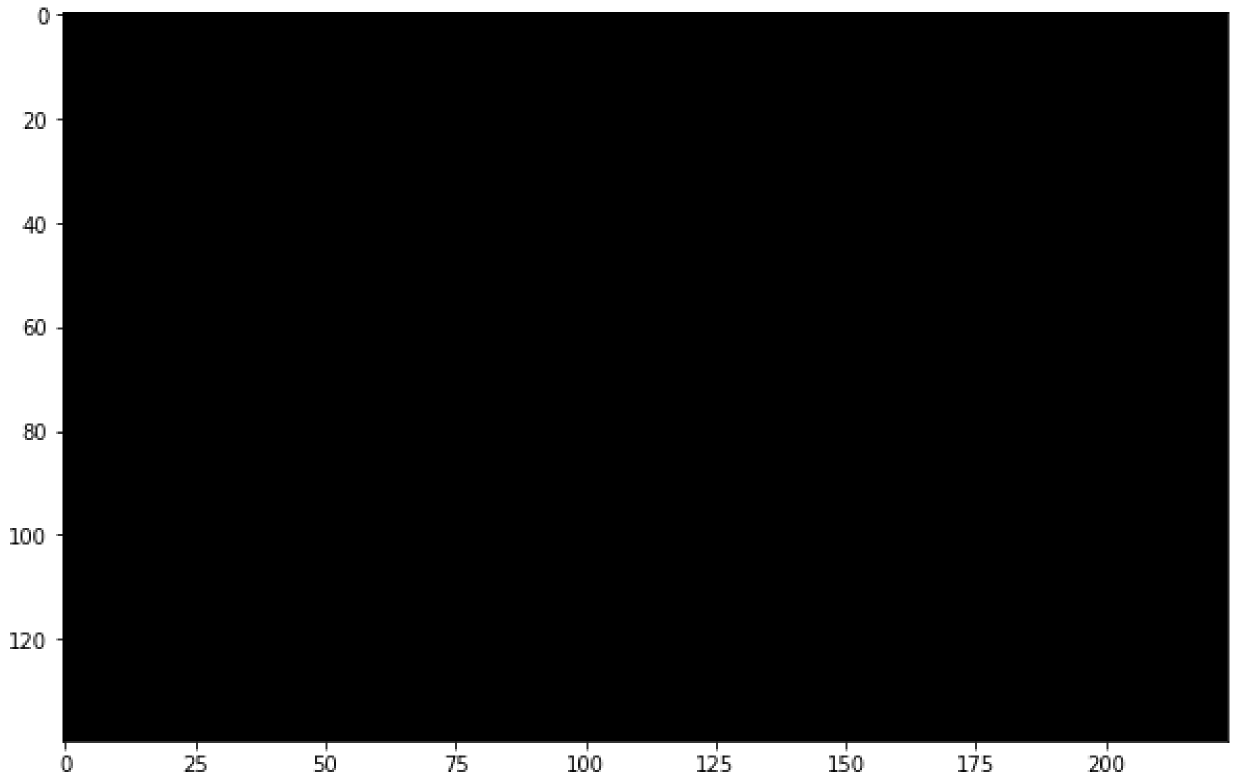
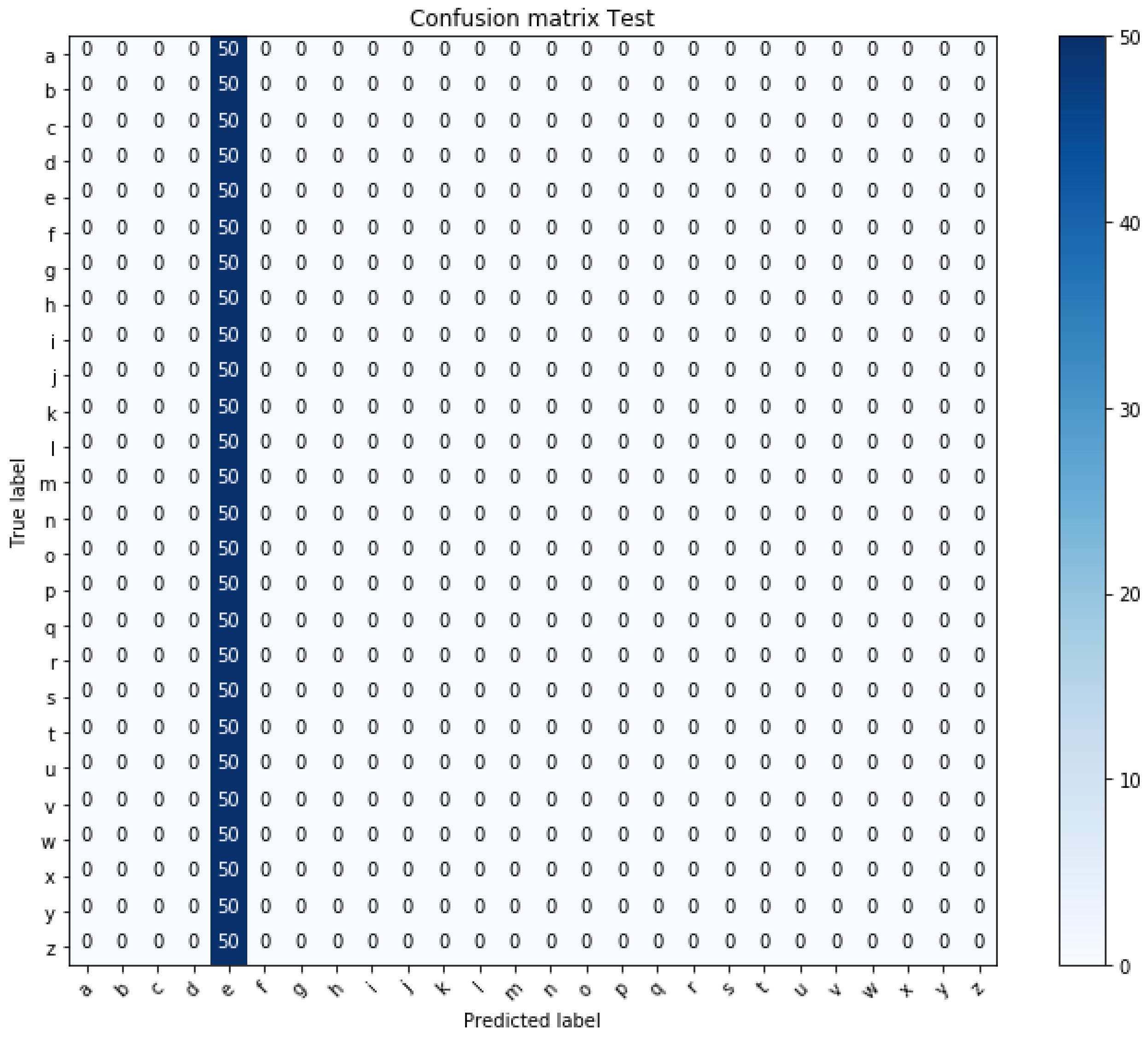
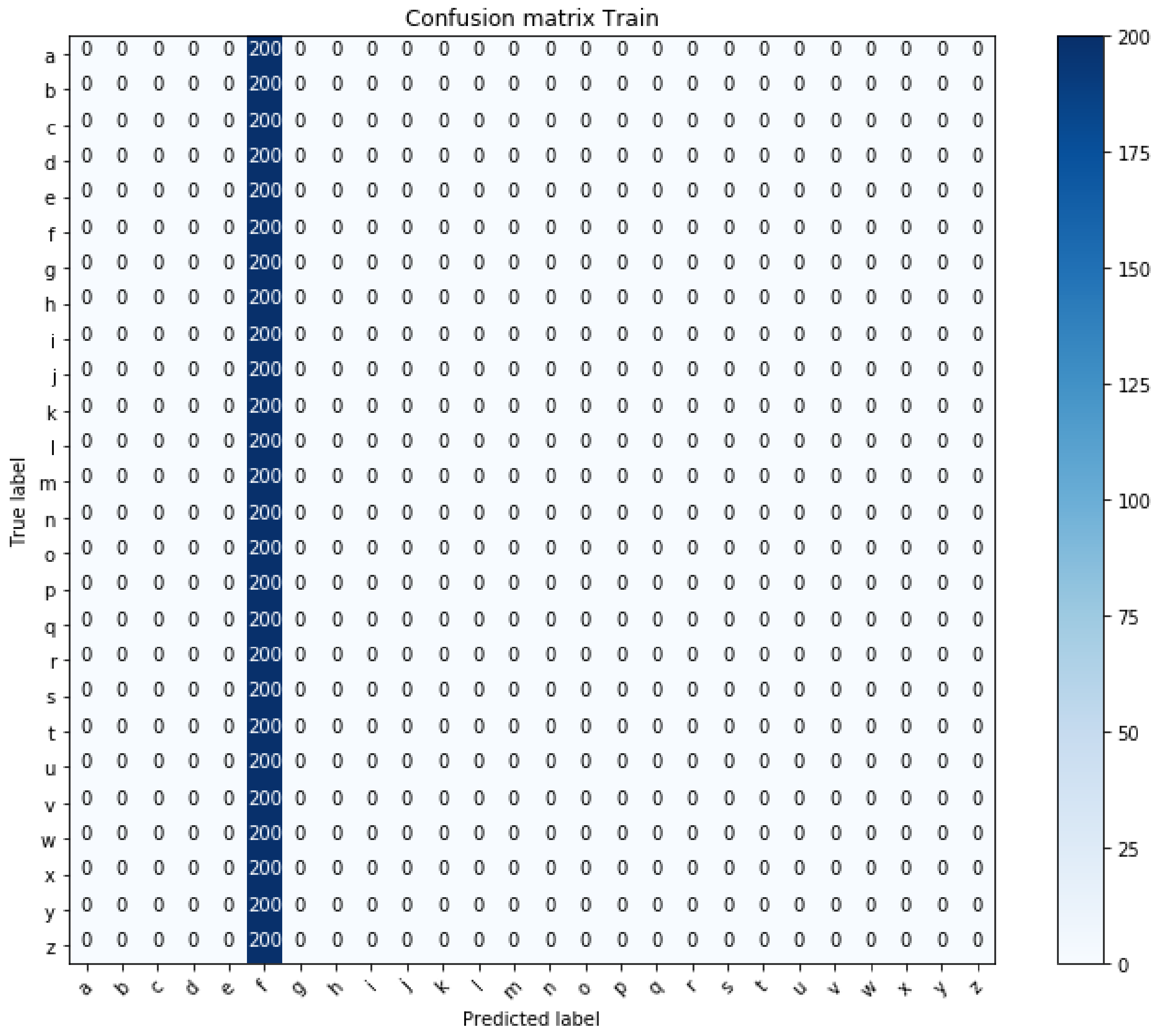


Figure:First layer weights too small therefore, all is black.



	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
g	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
j	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
w	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
y	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
z	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
g	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
j	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
w	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
y	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
z	0	0	0	0	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0