Assignment 1
Neural Networks and Deep Learning
CSCI 5922
Fall 2017

Bo Cao
boca7588@colorado.edu

All code are in “src” directory, in which there is a README.md file giving instructions on running the code.

Part 1

(1a) Report the values of w1, w2, and b.

Me:

w1:	-2.04424259514
w2:	3.99686016866
b:	-0.924290811868

(1b) What function or method did you use to find the least-squares solution?

Me:

Refer to https://jonathantemplin.com/files/regression/ersh8320f07/ersh8320f07_06.pdf

I calculated w1, w2 by:

$$w1 = (\Sigma {x_2}^2 \Sigma x_1 y - \Sigma x_1 x_2 \Sigma x_2 y) / (\Sigma {x_1}^2 \Sigma {x_2}^2 - (\Sigma x_1 x_2)^2)$$

$$w2 = (\Sigma {x_1}^2 \Sigma x_2 y - \Sigma x_1 x_2 \Sigma x_1 y) / (\Sigma {x_1}^2 \Sigma {x_2}^2 - (\Sigma x_1 x_2)^2)$$

$$b = \overline{Y} - w_1 \overline{X}_1 - w_2 \overline{X}_2$$

Notice that

$$\Sigma {x_1}^2 = \Sigma {X_1}^2 - (\Sigma X_1)^2 / N$$

$$\Sigma {x_2}^2 = \Sigma {X_2}^2 - (\Sigma X_2)^2 / N$$

$$\Sigma x_1 y = \Sigma X_1 Y - (\Sigma X_1 \Sigma Y) / N$$

$$\Sigma x_2 y = \Sigma X_2 Y - (\Sigma X_2 \Sigma Y) / N$$

$$\Sigma x_1 x_2 = \Sigma X_1 X_2 - (\Sigma X_1)(\Sigma X_2) / N$$

$\overline{Y}$ is the mean of all values in y, $\overline{X}_1$ is the mean of all values in X1, $\overline{X}_2$ is the mean of all values in X2.

X1 is the array containing all values of x1 in the dataset, X2 is the array containing all values of x2 in the dataset.

N is the size of the dataset, here N = 100.

The result is identical to the one from the method LinearRegression() from sklearn.

Part 2

(2a) Report the values of w1, w2, and b.

Me:

I used different settings to test the performance of the neural network, the settings are as followed:

Learning rate	Online (batch_size = 1)	Minibatch (batch_size = 5)	Minibatch (batch_size = 25)	Batch (batch_size = 100)
0.001	w1: -2.03605270061 w2: 3.99145124293 b: -0.925464016475 num of epoch: **1184**	w1: -1.96734598537 w2: 3.98887913637 b: -0.930133625922 num of epoch: **608**	w1: 0.410170068484 w2: 1.49599270416 b: -0.907766107214 num of epoch: **804**	w1: 1.31249432078 w2: -0.358748910851 b: -0.17465909398 num of epoch: **6208**
0.01	w1: -2.04407267827 w2: 4.00259226189 b: -0.923152645666 num of epoch: **448**	w1: -1.97027012833 w2: 3.99404850183 b: -0.926632208024 num of epoch: **3964**	w1: 0.869574225061 w2: 0.682107560327 b: -0.76415180406 num of epoch: **84**	w1: 0.333167689555 w2: -0.595008515391 b: 0.344045581052 num of epoch: **744**

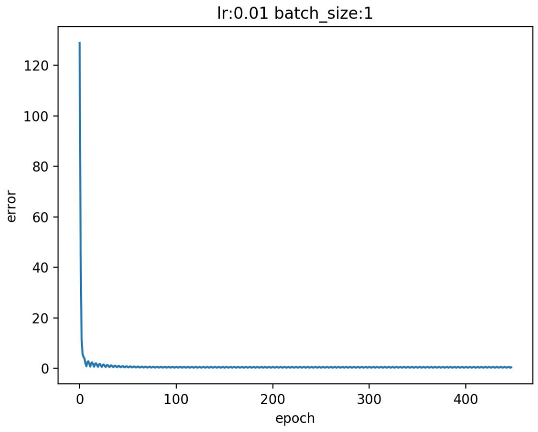
Table 1

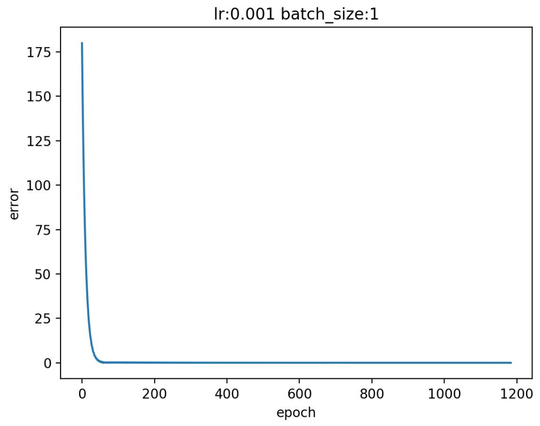
(2b) What settings worked well for you: online vs. batch vs. minibatch? what step size? how did you decide to terminate?
Me:
See table 1. The setting of (learning rate = 0.01 and online training) worked well as the weights is closer to the ideal solution from part1 with fewest epochs number of 448. When using minibatch (batch_size = 25) or batch training, the weights did not converge to the solution similar to part1 so I did not consider these two columns.

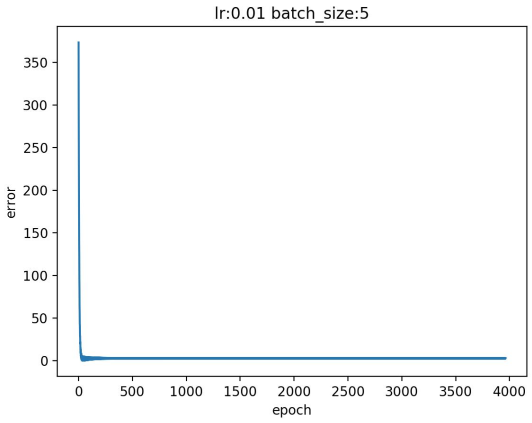
I used two ways to terminate: 1) when the epoch number is greater 25000, or 2) when the error on validation set starts to increase (early stopping). In method 2, the whole training set was splitted into training set and validation set with 3:1 proportion. Training set was used to update weights(w1, w2 and b) while validation set was used to calculate error compared to previous one. One guess of the reason why batch_size = 25 or 100 did not converge to the ideal solution could be that the error on validation set could jitter but not actually start to increase, but this code stops training as long as the error on validation set starts to increase.

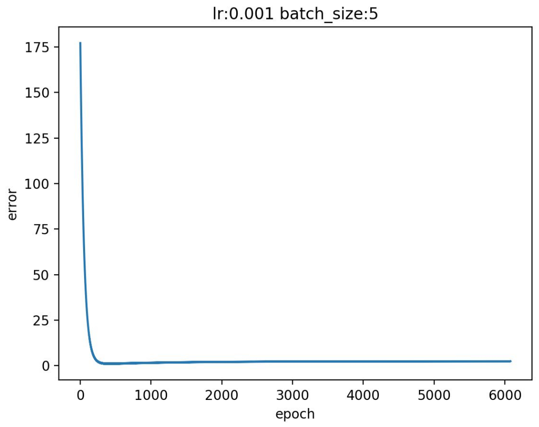
After each epoch, the error on the entire dataset, as well as the distance between the current weights to the correct weights were calculated. Distance was calculated via Euclidean distance.

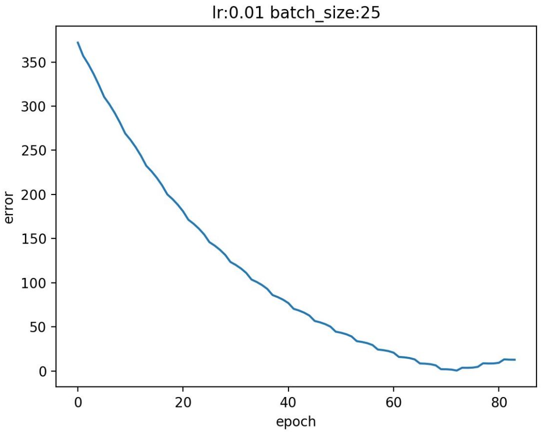
(2c) Make a graph of error on the entire data set as a function of epoch. An epoch is a complete sweep through all the data.
Me:
Graph of **error**:

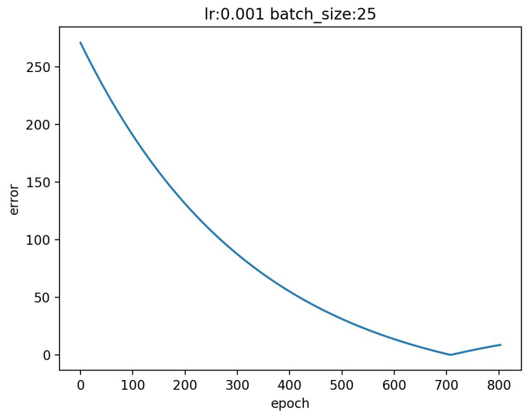

lr:0.01 batch_size:1

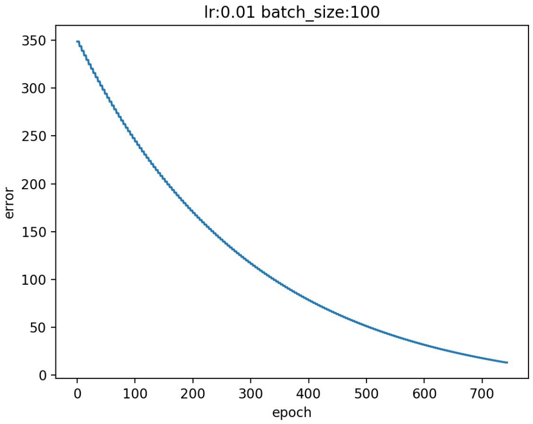

lr:0.001 batch_size:1

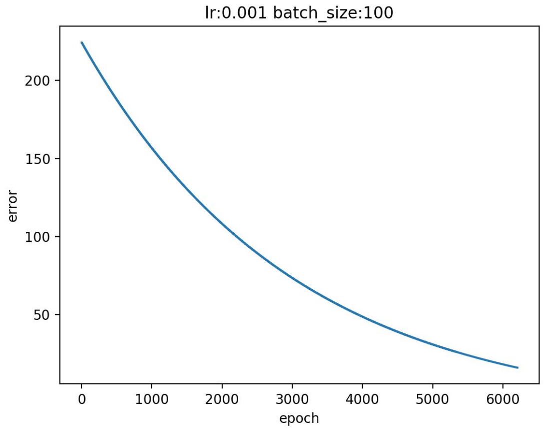

lr:0.01 batch_size:5


lr:0.001 batch_size:5

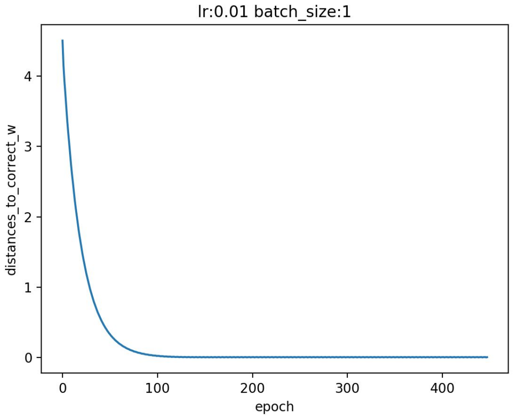


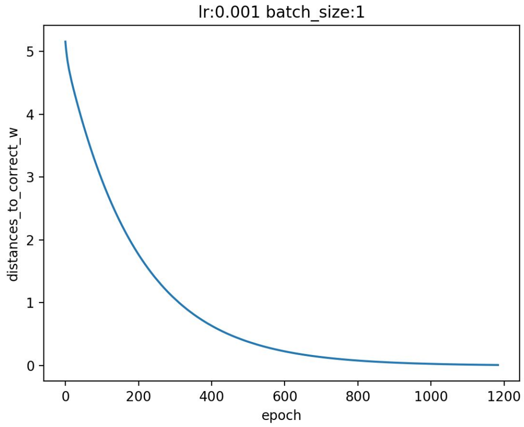


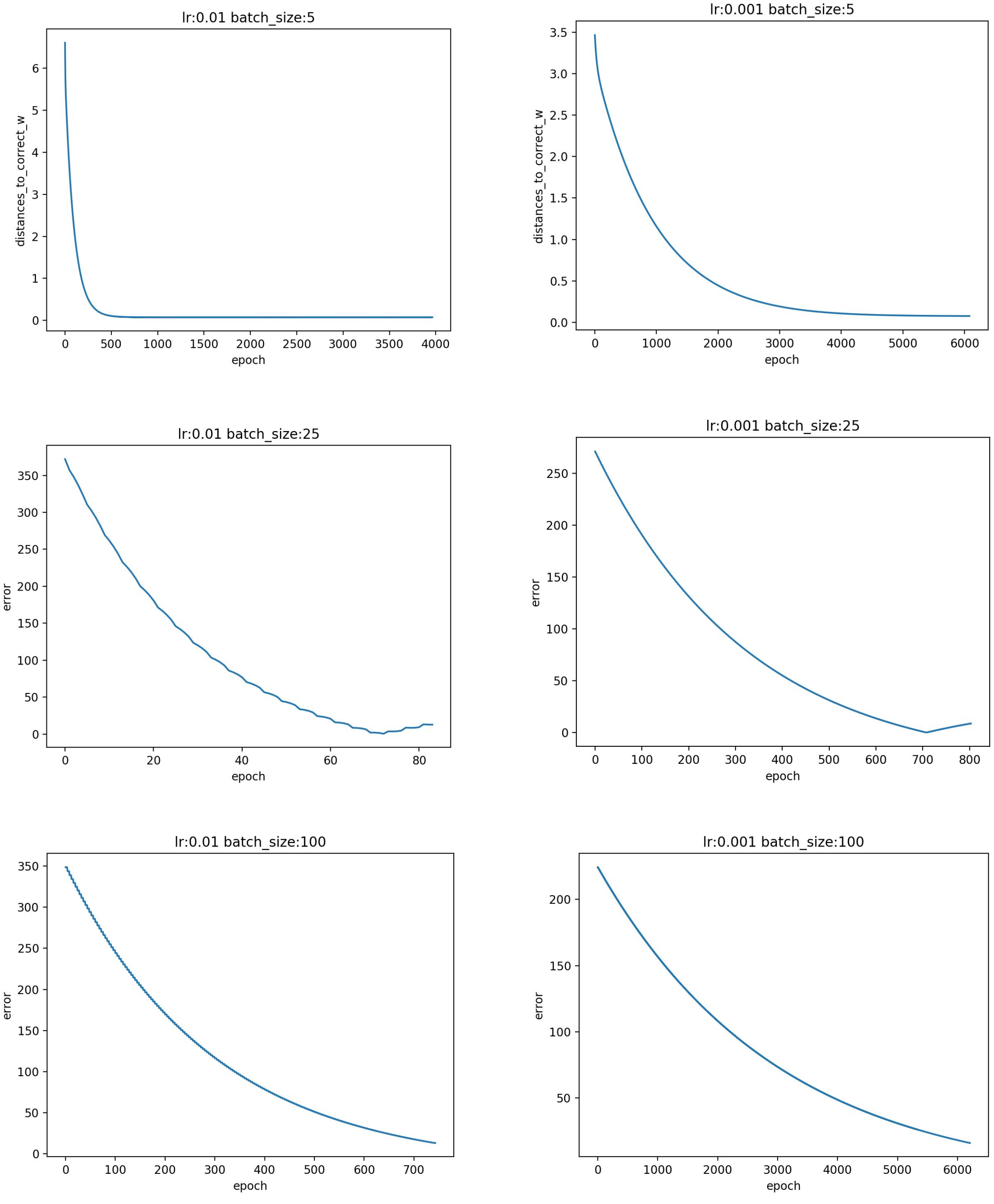




Graph of **distance** to exact solution:






lr:0.01 batch_size:5

lr:0.001 batch_size:5

lr:0.01 batch_size:25

lr:0.001 batch_size:25

lr:0.01 batch_size:100

lr:0.001 batch_size:100

Part 3

I used sigmoid as the activation function and 0.5 as the threshold, that is to say, when output from activation function is greater than 0.5, then the neuron outputs 1, otherwize outputs 0.

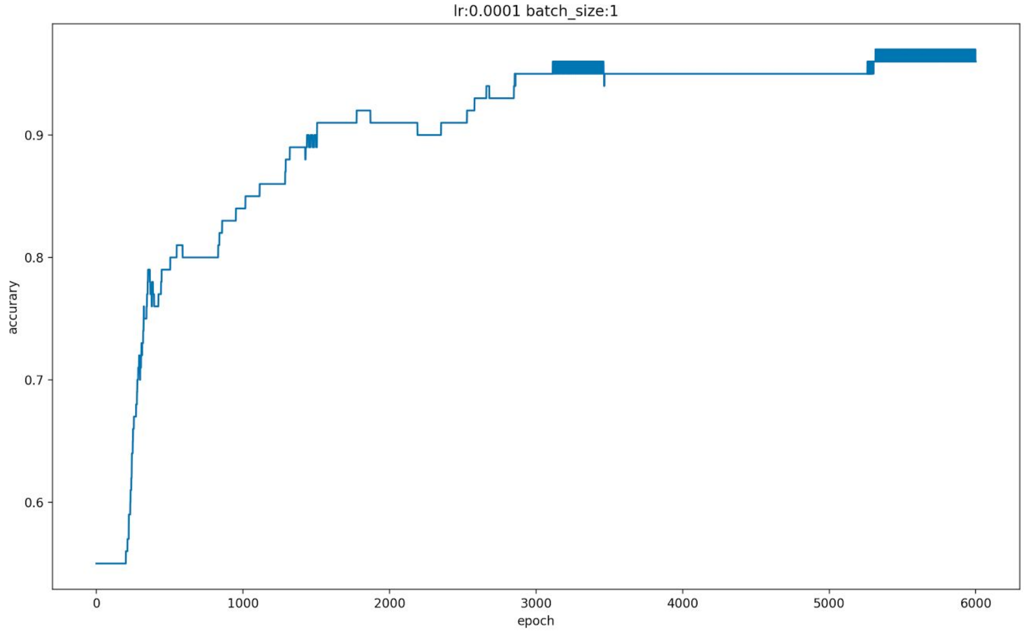
(3a) Report the values of coefficients w1, w2, and b.

Me:

Result: w1: -0.528361724132 w2: 0.871620757664 b: -0.160139904496

(3b) Make a graph of the accuracy (% correct classification) on the training set as a function of epoch.

Me:



We could see from the graph that when epoch is among 5000 to 6000 the accuracy jitters.

Part 4

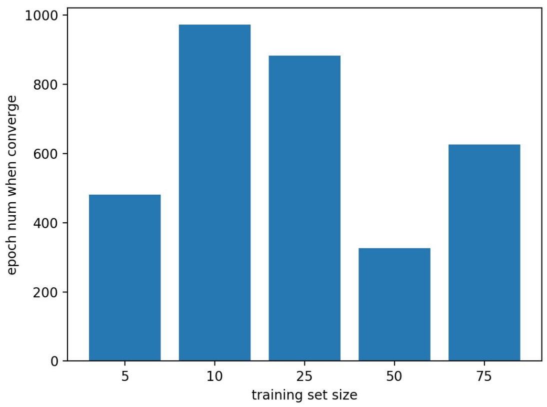
(4a) How does performance on the test set vary with the amount of training data? Make a bar graph showing performance for each of the different training set sizes.

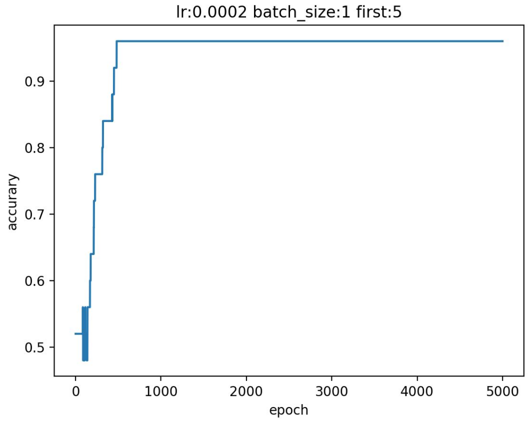
Me:

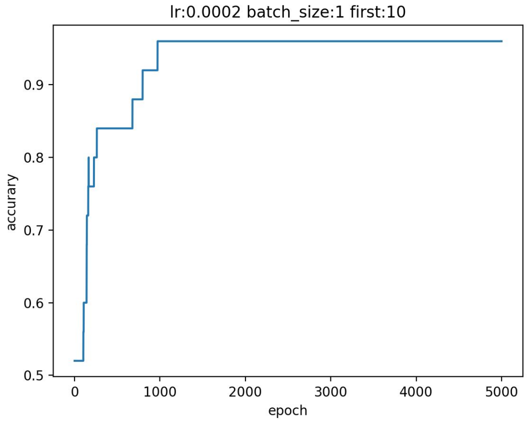
w1: -0.52760525866 w2: 0.77147356485 b: -0.10391519966

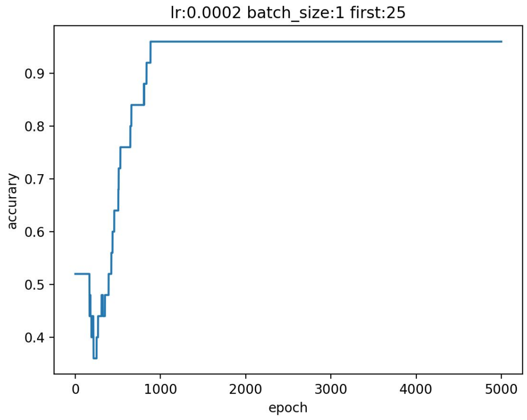
num of epoch: 5001

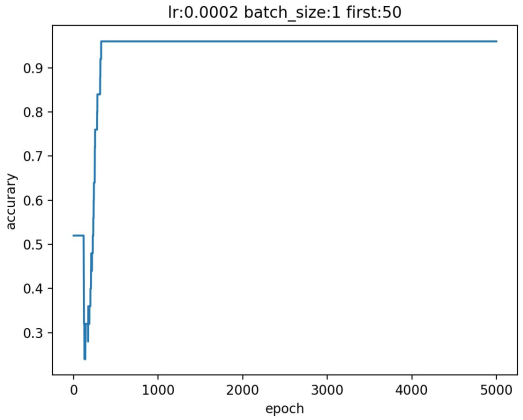
Based on the experiment, accuracy will converge to 96%, to show the performance on the test set, we record the epoch number when accuracy increases to 96%.

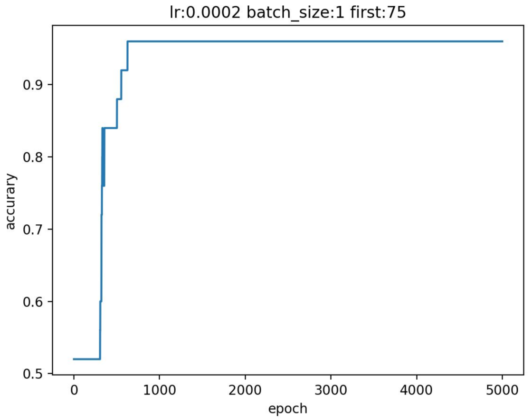

epoch num when converge
training set size


lr:0.0002 batch_size:1 first:5
accurary
epoch


lr:0.0002 batch_size:1 first:10
accurary
epoch


lr:0.0002 batch_size:1 first:25
accurary
epoch


lr:0.0002 batch_size:1 first:50
accurary
epoch


lr:0.0002 batch_size:1 first:75
accurary
epoch