DD2424 Assignment 1

Wei Wang
wewang@kth.se
Royal Institute of Technology

DD2424 Deep Learning in Data Science
Assignment Report

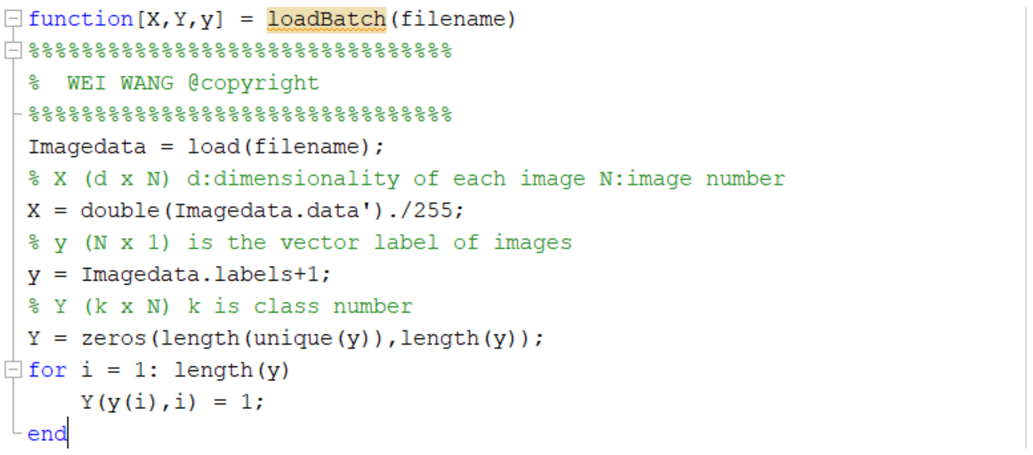
Introduction

The of object of this assignment is to train and test a one layer network with multiple outputs to classify images from the CIFAR-10 dataset. In this assignment we will train the network using mini-batch gradient descent applied to a cost function that computes the cross-entropy loss of the classier applied to the labelled training data and an L2 regularization term on the weight matrix.

Functions

In order to successfully manage to write the functions to analytically compute the gradient. Here are several subfunctions I have to write at first:

1. ***function [X, Y, y] = LoadBatch(filename)***
   Load the filename at first and then transfered the images to double type and then managed y by add 1 to all Imagedata.Labels and finally generate one-hot representation by iteration:

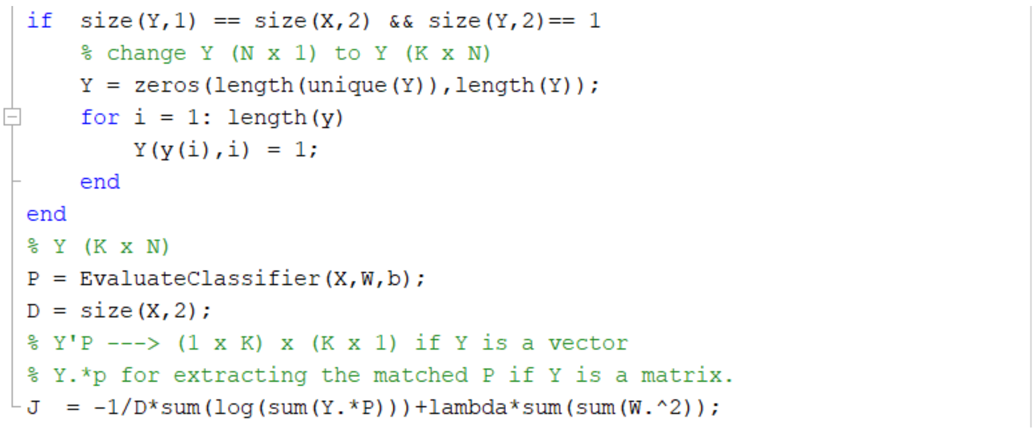


```
function[X,Y,y] = loadBatch(filename)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  WEI WANG @copyright
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Imagedata = load(filename);
% X (d x N) d:dimensionality of each image N:image number
X = double(Imagedata.data')./255;
% y (N x 1) is the vector label of images
y = Imagedata.labels+1;
% Y (k x N) k is class number
Y = zeros(length(unique(y)),length(y));
for i = 1: length(y)
    Y(y(i),i) = 1;
end
```

Figure 1: code of *LoadBatch*() function

2. ***function P = EvaluateClassifier(X, W, b)***
   calculate the P (probability) by the function $s = Wx+b$ and $P = SOFTMAX(S)$. Using the function *repmat*() to add b to Wx with matched dimension.

3. ***function J = ComputeCost(X, Y, W, b, lambda)***
   Judge whether the Y is vector or one-hot matrix at first. If the Y is vector type, then transfer it to one-hot presentation. Using the equation 5 mentioned in the

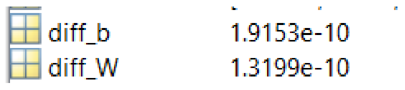


```
if  size(Y,1) == size(X,2) && size(Y,2)== 1
    % change Y (N x 1) to Y (K x N)
    Y = zeros(length(unique(Y)),length(Y));
    for i = 1: length(y)
        Y(y(i),i) = 1;
    end
end
% Y (K x N)
P = EvaluateClassifier(X,W,b);
D = size(X,2);
% Y'P ---> (1 x K) x (K x 1) if Y is a vector
% Y.*p for extracting the matched P if Y is a matrix.
J  = -1/D*sum(log(sum(Y.*P)))+lambda*sum(sum(W.^2));
```

Instruction to calculate the cost. Here the $Y^T P$ should be $Y. * P$ in Matlab for matching the multi-dimension of Y.

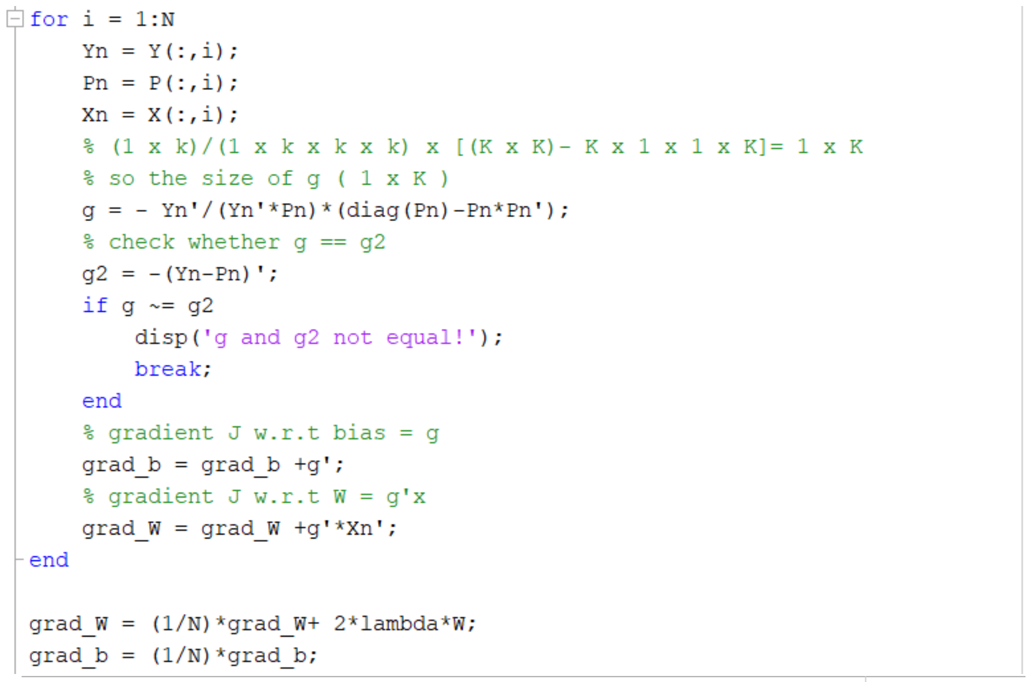
4. ***function acc = ComputeAccuracy(X, y, W, b)***
   For calculating the accuracy of predictions. I divide the right matched classes by the whole prediction number:$acc = sum(Index == y')/length(y)$; I can also calculate the accuracy by using onehot Y .* onehot outcome and then sum up.

5. ***function [grad_W, grad_b] = ComputeGradients(X, Y, P, W, lambda)***
   As we know the pipeline of this linear classifier is "linear scoring function + SOFTMAX + cross-entropy loss + Regularization". For calculating the gradient l with respect to W, b, I followed the instruction in Lecture 3 through Backpropagation Method. I first set all entries in grad_w and grad_b to be zero. Then, for each iteration of (X,Y), I update the gradient by these equation:$\frac{\partial L}{\partial b} += g$ and $\frac{\partial L}{\partial W} += g^T X^T$. Here the $a = -\frac{y^T}{y^T P}(diag(P) - pp^T) = -(y - P)^T$ .Then I divide them by $|D^{(t)}|$ and finally add the gradient for the regulation term:$\frac{\partial J}{\partial W} = \frac{\partial L}{\partial W} + 2\lambda W$. The code is shown as follows dimension of Y. Here I also compared g and g2 to check the outcome of my code. After calculating gradient component. I need compare the gradient component with the outcome of $ComputeGradsNumslow()$ function. The relative error is calculated and displayed in assignment1.m . In workplace, we can see the relative error between $ComputeGradients()$ and $ComputeGradsNumslow()$ is much smaller than $1e^-6$.

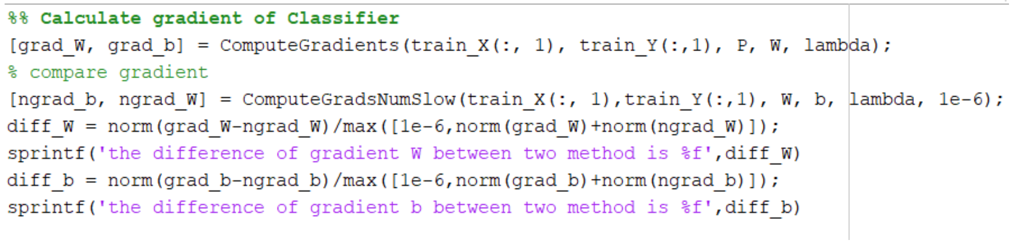


```
diff_b      1.9153e-10
diff_W      1.3199e-10
```

Figure 2: relative error between two functions



```
for i = 1:N
    Yn = Y(:,i);
    Pn = P(:,i);
    Xn = X(:,i);
    % (1 x k)/(1 x k x k x k) x [(K x K)- K x 1 x 1 x K]= 1 x K
    % so the size of g ( 1 x K )
    g = - Yn'/(Yn'*Pn)*(diag(Pn)-Pn*Pn');
    % check whether g == g2
    g2 = -(Yn-Pn)';
    if g ~= g2
        disp('g and g2 not equal!');
        break;
    end
    % gradient J w.r.t bias = g
    grad_b = grad_b +g';
    % gradient J w.r.t W = g'x
    grad_W = grad_W +g'*Xn';
end

grad_W = (1/N)*grad_W+ 2*lambda*W;
grad_b = (1/N)*grad_b;
```



```
%% Calculate gradient of Classifier
[grad_W, grad_b] = ComputeGradients(train_X(:, 1), train_Y(:,1), P, W, lambda);
% compare gradient
[ngrad_b, ngrad_W] = ComputeGradsNumSlow(train_X(:, 1),train_Y(:,1), W, b, lambda, 1e-6);
diff_W = norm(grad_W-ngrad_W)/max([1e-6,norm(grad_W)+norm(ngrad_W)]);
sprintf('the difference of gradient W between two method is %f',diff_W)
diff_b = norm(grad_b-ngrad_b)/max([1e-6,norm(grad_b)+norm(ngrad_b)]);
sprintf('the difference of gradient b between two method is %f',diff_b)
```

6. ***function [Wstar, bstar] = MiniBatchGD(X, Y, GDparams, W, b, lambda)***
   To calculate the gradient component by mini-batch gradient descent algorithm, I followed the snippet of code of assignment instruction and equation 8 and equation 9. For recording loss and accuracy more easily, I didn't use n_epoch in $MiniBatch()$. Instead, I ran the epochs in assignment1.m

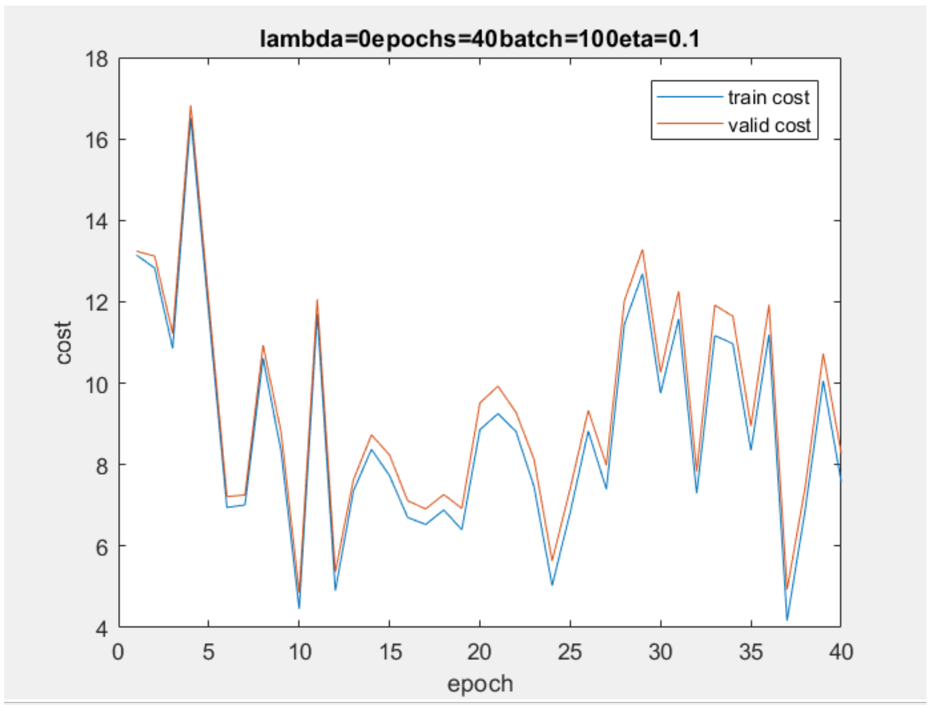
Results

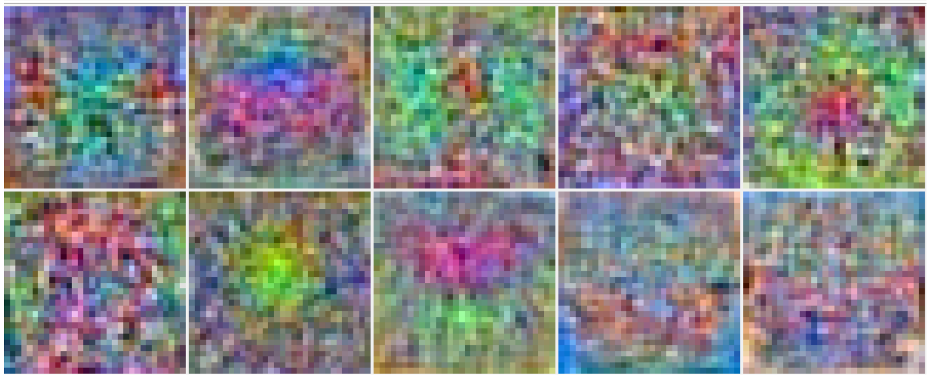
The graphs of total loss and cost function on the training data and the validation data after each epoch of the mini-batch gradient descent are shown as follows:

- ***lambda = 0, n_epochs = 40, n_batch =100,eta = 1***
  1. We can see that the loss curve of train and validation is not "stable". These curves "shake" too much. The cost go up and down strongly after each epoch. Because the learning rate eta is set to be 1. the eta decide how "big" one "step" should be while conducting the mini-batch gradient descent algorithm. When

the learning rate is too large, finding the local minimum become hard because it's so easy to "step over".

2. Also, the graphs of Weight matrix is blur and noisy, which means the intensity of W is mussy. The W doesn't learned the information from image well.

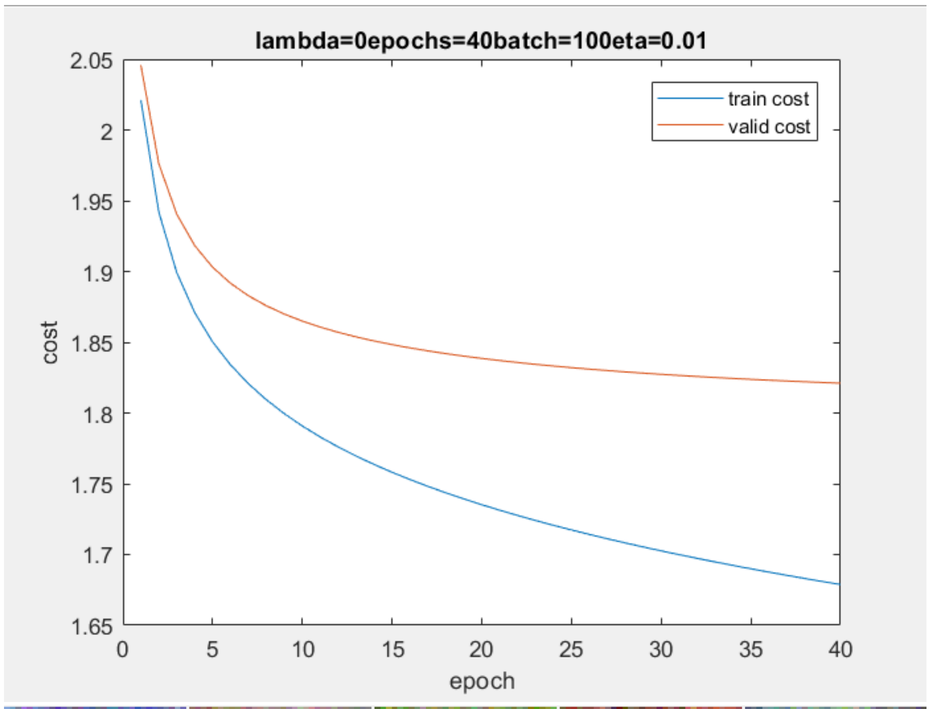


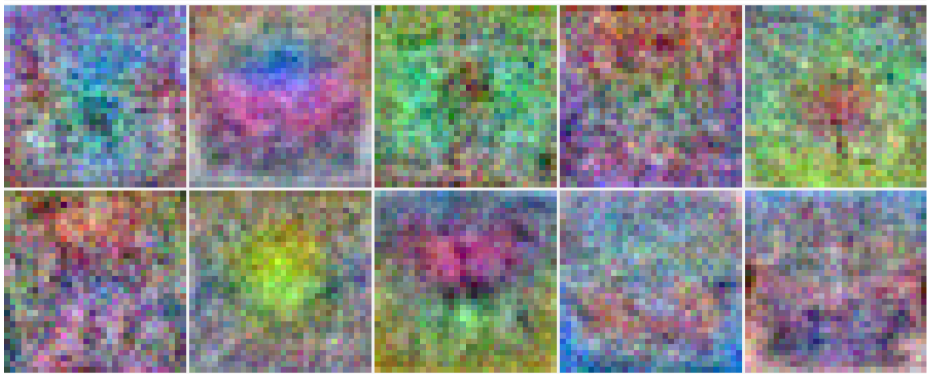


- ***lambda = 0, n_epochs = 40, n_batch =100,eta = 0.1***

  1. After change the eta to 0.1. The curve of cost become smooth since the each "step" after every iteration become smaller and it's easier to find the local minimum. Using eta = 0.1 improve the out come of classifier and enjoys a lower cost both training set and validation set.

  2. The graph of weight matrix becomes "clearer" and less noisy. The intensity of Matrix seems more smooth than previous one. That is because the classifier has already reached more higher accuracy and W has already learned some information from training images.

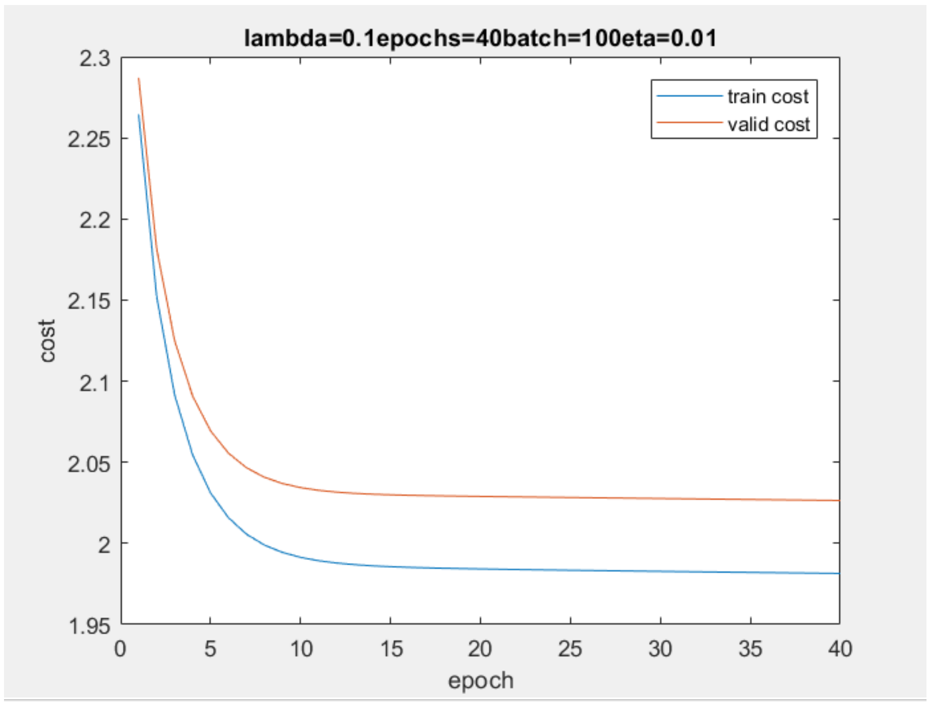


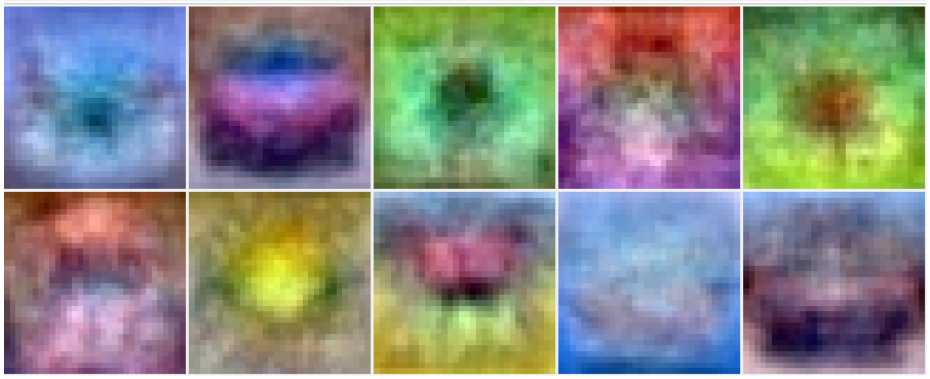


- ***lambda = 0.1, n_epochs = 40, n_batch =100,eta = 0.1***

  1. As we can see, after adding regulation term (lambda = 0.1), the curve converge faster. It reached local minimum using less time and the curve is also smooth and stable since we used the same suitable learning rate(eta = 0.1). But the result seems not improved so much somehow. The training curve converges around 1.7, a bit worse than previous one.

  2. The graphs of W became even "clearer" and more "smooth". we can even see the outline of "car" in graph 2 and "horse" in graph 8. By adding regularization, the classifier reduced the complexity and makes W with more sparsity.

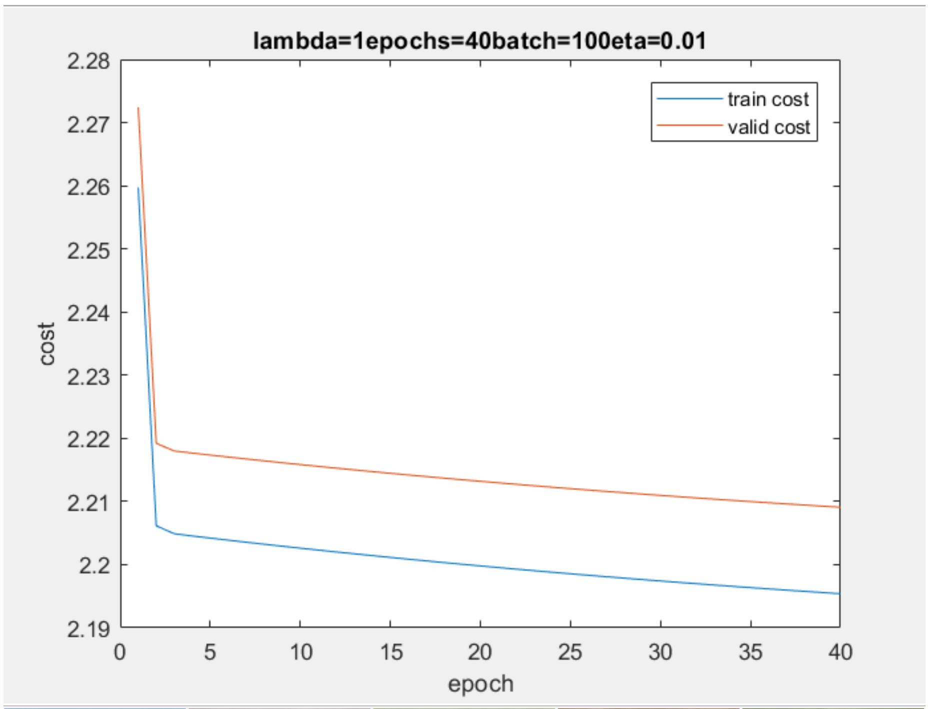


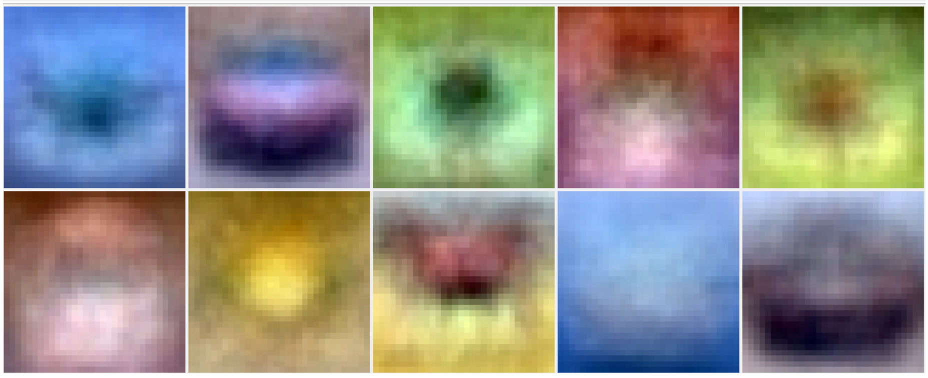


- ***lambda = 1, n_epochs = 40, n_batch =100,eta = 0.1***

  1. when we enlarge lambda, the regularization term has larger weight and larger influence.The curve converge much faster with less time. The difference between training curve and validation curve became bigger. The loss became larger.

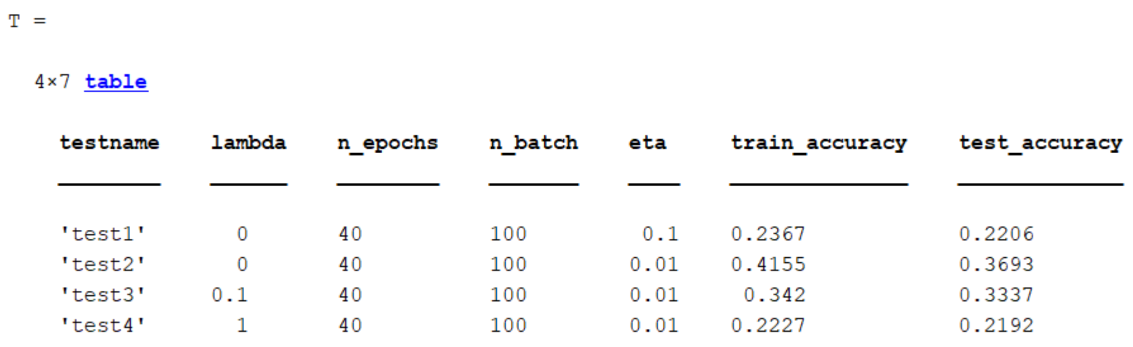
  2. The color of graphs of W became lighter. It shows that the average of intensity of W became smaller. The average of histogram shift to smaller number. The W become more sparse. It shows that the complexity of classifier is reduced. It can prevent over-fitting and improve the generalization ability and reduce the variance of classifier. But the error is increased.





Test accuracy

To record the test accuracy I re-ran the code and record the accuracy in the table below. This table is the screenshot from command window of Matlab. From the table,

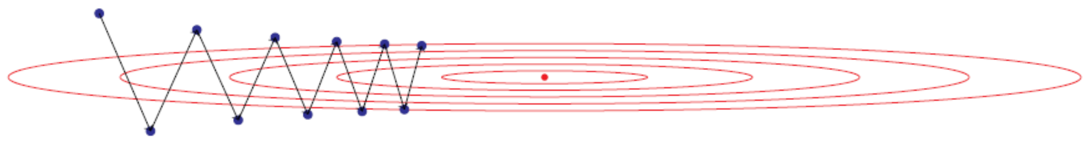


```
T =

  4×7 table

    testname    lambda    n_epochs    n_batch    eta     train_accuracy    test_accuracy
    ________    ______    ________    _______    ____    ______________    _____________

    'test1'        0         40         100       0.1    0.2367            0.2206
    'test2'        0         40         100      0.01    0.4155            0.3693
    'test3'      0.1         40         100      0.01     0.342            0.3337
    'test4'        1         40         100      0.01    0.2227            0.2192
```

we could see that the second group reached the highest accuracy: around 41.5% for training accuracy and around 36.9% for test accuracy.

Conclusion

Learning rate can decide how ”big” one step is after each iteration. If the learning rate is too large, the classifier will often ”step over” and the learning curve will be shaking. If the learning rate is too small the convergence time will be longer. When we reduce the eta to 0.1 the curve become smooth.



The regularization term aims to ”regularize” the classifier. It aims to reduce the complexity of classifier and increase sparsity in W. It can prevent over-fitting and reduce variance efficiently, but it also increase the error when the regulation term become larger. When we enlarge the lambda, the classifier converge faster since the ”complexity” of W is reduced. The accuracy decrease when lambda is too large. The W's histogram shift to zero side and the intensity of W become more smooth.