ASSIGNMENT 2

In this assignment, we train and test a 2 layer network with multiple outputs to classify images from CIFAR -10 data set. We have one hidden layer of 50 nodes. We also have L2 Regularisation term. The output layer consists of 10 nodes, the total number of unique class labels of the data set. We use Relu as activation function in the hidden layer and Softmax as activation function in the output layer. The probability from Softmax decides the predicted class label if it has the maximum probability. We also use cyclic learning rates with mini batch gradient descent and finally do grid search to identify best regularisation lambda.

Forward Propogation

We initialise data batch 1 for training and data batch 2 for validation. The training and validation sets are normalised by mean standard deviation normalisation. Each batch contains n=10000 images

d- No of features or no of image pixels=3072
n-no of input samples
c-no of class labels=10
X- Input training data of the form (d x n)
Y- one-hot representation of the label for each image. of form (c x n)

Then we initialise , the weight and bias. Since it is a two layer network we have W1,W2,b1,b2. we initialise the entry with Gaussian random values with zero mean and standard deviation .01 in W1,W2 and b1,b2.

No of nodes in hidden layer=50

W1 - weight matrix of 1st layer (50 x d)
W2 - weight matrix of 2nd layer (c x 50)
b1 -bias term of 1st layer (50 x 1)
b2- bias term of 2 nd layer (c x 1)

Then the images are evaluated and probabilities are predicted for the images .
$s_1 = W_1 x + b_1$
h = max(0, s_1) —Relu activation function in hidden layer (50 x n)
$s = W_2 h + b_2$
p = SOFTMAX(s) (c x n)

Here the predicted class corresponds to the label with the highest probability.

Then we calculate the cross entropy loss function with L2 regularisation on the weights matrix

$$J(\mathcal{D}, \lambda, \mathbf{\Theta}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} l_{\text{cross}}(\mathbf{x}_i, y_i, \mathbf{\Theta}) + \lambda \sum_{l=1}^{2} \sum_{i,j} W_{l,ij}^2$$

Backward Propagation

Then the gradients are computed by analytical method for the weights and bias of the output layer as well as the weights and bias of the hidden layer.

G=-(Y-P)

grad_W2n(GxH.T)/n+2*lambda*W2

grad_W1=(GxX.T)/n+2*lambda*W1

Then we try to check if the gradients that are computed are correct. So we compare the gradients from analytical method to the gradients from numerical method.

Initially we take one image and 20 dimensions without regularisation term from training data set and there are less absolute or relative errors above 1 e-6.

```
w1: 100.0% of absolute errors below 1e-6
b1: 100.0% of absolute errors below 1e-6
w2: 100.0% of absolute errors below 1e-6
b2: 100.0% of absolute errors below 1e-6
w1: 98.4% of relative errors below 1e-6
b1: 98.0% of relative errors below 1e-6
w2: 100.0% of relative errors below 1e-6
b2: 100.0% of relative errors below 1e-6
```

Similarly we calculate relative and absolute error for gradients with 5 images and using all dimensions from training set.

```
w1: 100.0% of absolute errors below 1e-6
b1: 100.0% of absolute errors below 1e-6
w2: 100.0% of absolute errors below 1e-6
b2: 100.0% of absolute errors below 1e-6
w1: 74.84635416666666% of relative errors below 1e-6
b1: 98.0% of relative errors below 1e-6
w2: 84.0% of relative errors below 1e-6
b2: 100.0% of relative errors below 1e-6
```

Then we train the network with cyclical learning rates in the same mini batch gradient descent fashion. So it differs from Vanilla mini batch gradient descent which uses fixed learning rates. The main idea of cyclical learning rates is that during training the learning rate is periodically changed in a systematic fashion from a small value to a large one and then from this large value back to the small value .One complete cycle will take $2n_s$ update steps, where n_s is known as the stepsize. Thus When $t = 2ln_s$ then $\eta_t = \eta_{min}$ and when $t = (2l+1)n_s$ then $\eta_t = \eta_{max}$.

at t th step, learning rate is calculated as

At iteration t of training if $2ln_s \leq t \leq (2l+1)n_s$ for some l $\{0, 1, 2, \ldots\}$ set

$$\eta_t = \eta_{\min} + \frac{t - 2ln_s}{n_s}(\eta_{\max} - \eta_{\min}) \qquad (1$$

if $(2l+1)n_s \leq t \leq 2(l+1)n_s$ for some $l \in \{0, 1, 2, \ldots\}$ set

$$\eta_t = \eta_{\max} - \frac{t - (2\,l+1)n_s}{n_s}(\eta_{\max} - \eta_{\min})$$

After each update step we have to save the metrics of accuracy and loss computed. One update step is calculated as when the entire is when the entire training set is traversed in batches, So we monitor the index of number of batches that has been traversed. Then the estimate W1,W2,b1 and b2 is updated with the following equations after each batch is processed.

$$W_k^{(t+1)} = W_k^{(t)} - \eta \left.\frac{\partial J(\mathcal{B}^{(t+1)}, \lambda, \boldsymbol{\Theta})}{\partial W_k}\right|_{\boldsymbol{\Theta}=\boldsymbol{\Theta}^{(t)}}$$

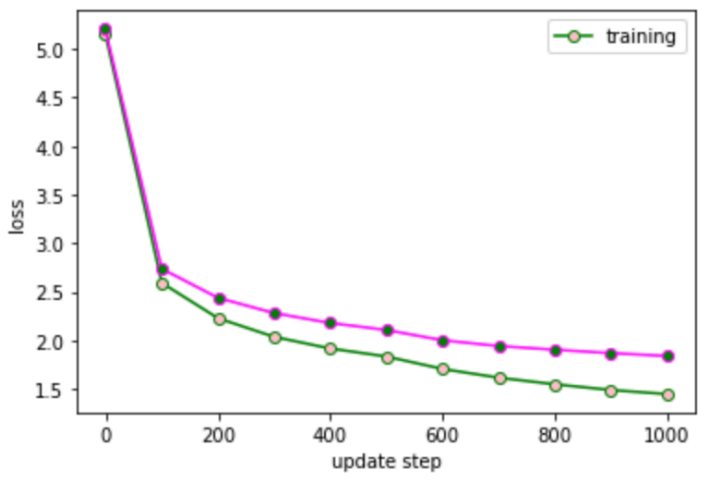
$$\mathbf{b}_k^{(t+1)} = \mathbf{b}_k^{(t)} - \eta \left.\frac{\partial J(\mathcal{B}^{(t+1)}, \lambda, \boldsymbol{\Theta})}{\partial \mathbf{b}_k}\right|_{\boldsymbol{\Theta}=\boldsymbol{\Theta}^{(t)}}$$

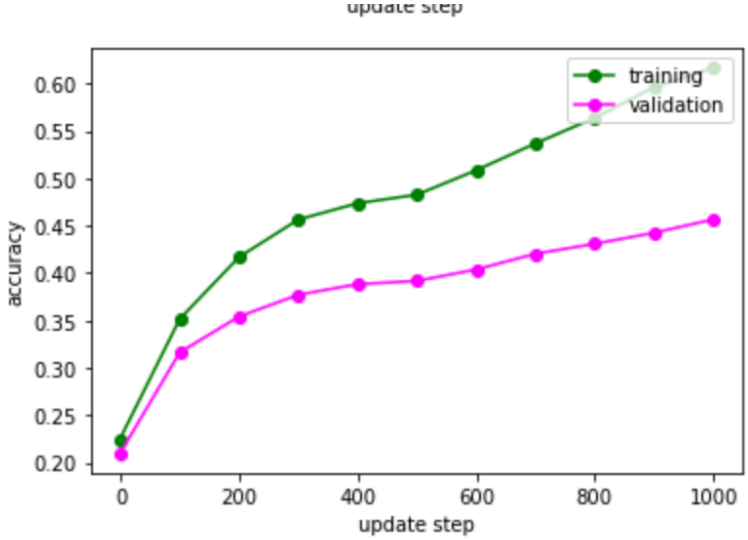
We train the network on a small amount of the training data (say 100 examples) with regularization turned off (λ=0) and check we overfit to the training data and get a very low loss on the training data after training for a suffcient number of epochs (~200) and with a reasonable η. Being able to achieve this indicates that gradient computations and mini-batch gradient descent algorithm are okay.

finishes epoch 200: loss=0.0007150910457046539 and accuracy=1.0 (training set)

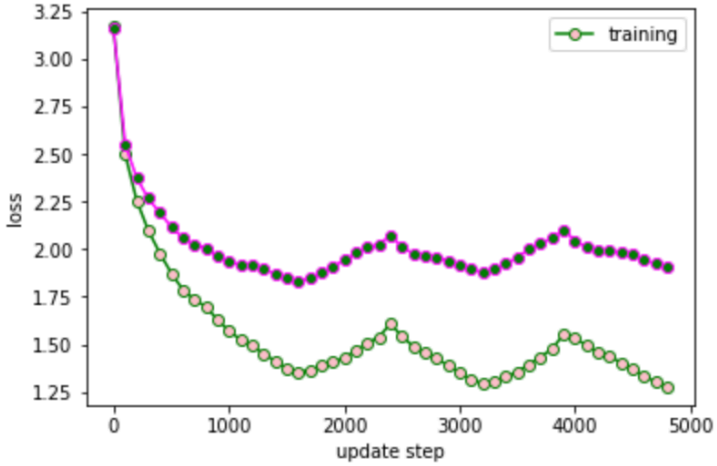
Then we train with eta min = 1e-5, eta max = 1e-1 and n s=500 and the batch size to 100, lambda=0.01. It is equivalent to 10 epochs = 1 cycles · 2 · 500 / (10.000/100)

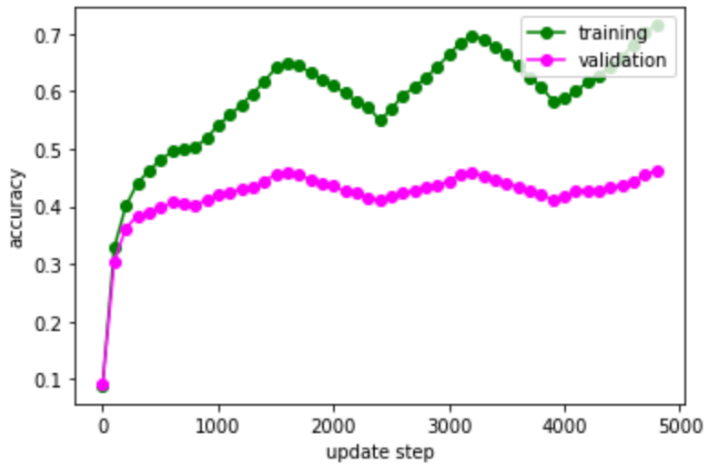
The learning curves for the above hyper parameter setting is plotted as a graph below.





Then we set eta min = 1e-5, eta max = 1e-1, lambda=.01 and batch size =800. it is equivalent to 3 cycles · 2 · 800 / (10.000/100) = 48 epochs. The result is plotted as graph. In the loss and accuracy plots you can clearly see how the loss and accuracy vary as the η_t varies.

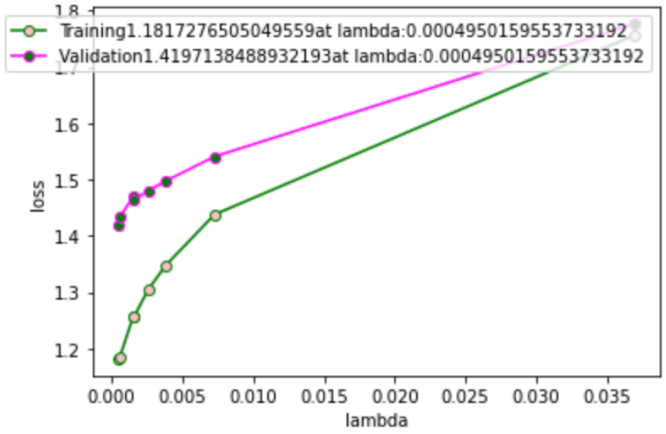


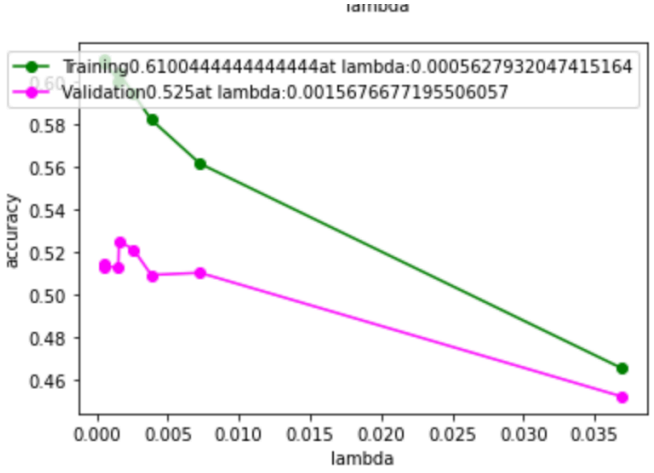


We have to perform Grid search in order to find the best value for lambda. We use of the most of the training data available (45000) for training from

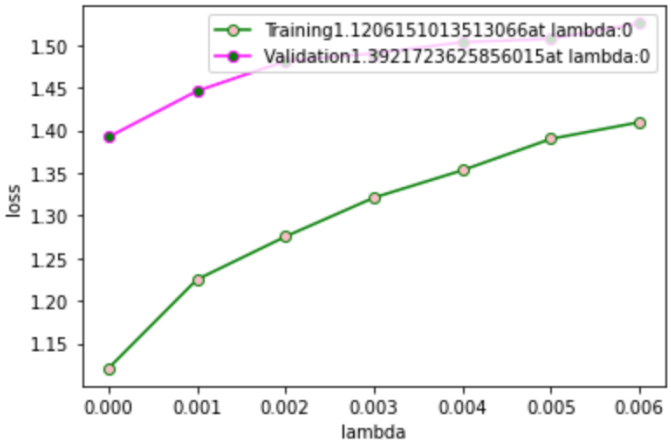
data_batch_1,data_batch_2,data_batch_3,data_batch_4,data_batch_5. Randomly 5000 images are used for validation. This is because more data and increasing the value of lambda are both forms of regularization. When you have less train- ing data you will need a higher lambda and when you have more training data you will need a lower lambda. Then we have list of 8 values from 10 to the power of random values with uniform sampling in range l min=-5 and l max=-1 . The list of lambdas used are [0.0004950159553733192,0.0005627932047415164,0.0015119336467640998, 0.0015676677195506057, 0.002576638574613588, 0.0038333321561566606, 0.007257005721594274, 0.03690557729213758]

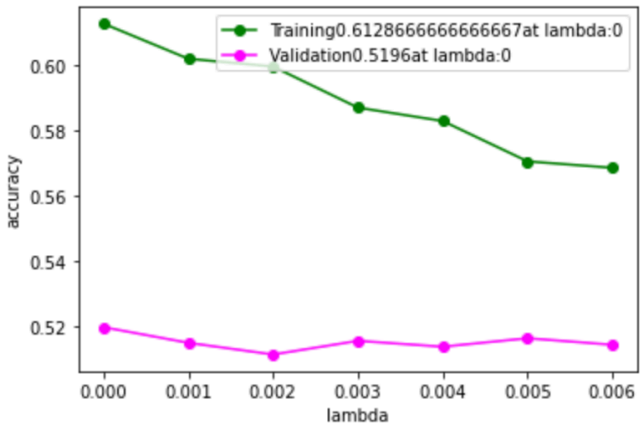
Then we have the grid search Cv to calculate the loss and accuracy for each value of lambda. eta_min=1e-5, eta_max=1e-1, step_size=900, n_batch=100, cycles=3 where the hyper parameter setting used. Then for each lambda, the corresponding loss and accuracy are plotted.



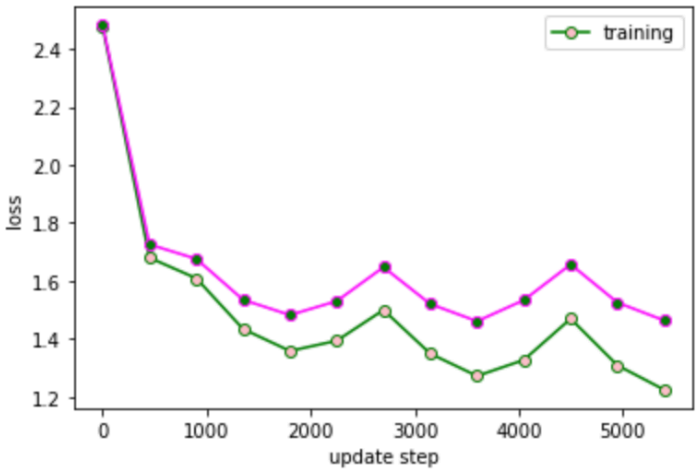


The loss is lower and accuracy is higher when lambda is low. Hence to do a fine search of lambda, used a list of [0,0.001,0.002,0.003,0.004,0.005,0.006]. For each value of lambdas , the obtained accuracy and loss are plotted.

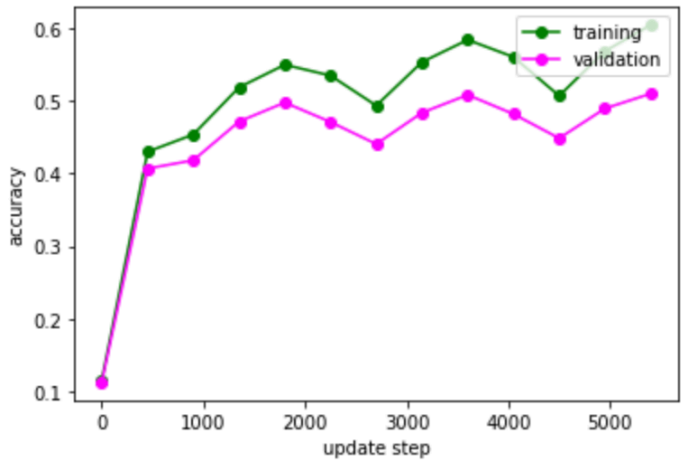




It is noted that at 0.001 regularisation the loss is lower and the accuracy is higher for both training and validation. So for the test set, we use 0.001 as regularisation . The the accuracy and loss are plotted after each epoch. An accuracy of 0.5102 is obtained for test set.


training
loss
update step
2.4
2.2
2.0
1.8
1.6
1.4
1.2
0
1000
2000
3000
4000
5000

update step


training
validation
accuracy
update step
0.6
0.5
0.4
0.3
0.2
0.1
0
1000
2000
3000
4000
5000