# DD2424 - Deep Learning in Data Science
# **Assignment 3 Report**

Ajinkya Khoche
khoche@kth.se

September 4, 2018

## 1. Objective

In Assignment 2, a two layer neural network was analyzed along with momentum. In Assignment 3, the following improvements were realized:

- A two layer neural network was extended to a general *'k' - layered network*. The number of hidden layers (and neurons per layer) can be set using *'hiddenLayerSizeList'* parameter
- Batch Normalization was implemented and it's effects analyzed.
- Exponential moving average of mean and variance for batch normalization was implemented.

The report is structured as follows. Section 2. compares analytical and numerical gradients for k-layer neural network. Section 3. includes the effect of batch normalization on training. Section 4. covers the coarse to fine search for best values of learning rate (eta) and regularization parameter (lambda). Section 5. analyzes the effect of batch normalization on speed of learning.

**NOTE**: Unless otherwise specified, the following hyper-parameters were used for training:

- Batch size: 100
- Momentum factor (rho): 0.9
- Exponential moving average parameter (alpha): 0.99

## 2. Checking Gradient Computation

The function 'CheckGradients()' compares analytical and numerical gradients. The analytical gradient is computed by 'ComputeGradients()' function. The numerical gradient has two variants. The faster but less accurate formula based on *finite difference method* is computed by calling 'ComputeGradientsNum()' function. The slower but accurate version based on *central difference method* is computed by calling 'ComputeGradientNumSlow()' function.

There are basically two methods of comparison. Method 1 checks if relative error is small by computing expression below:

$$error = \frac{|g_a - g_n|}{max(eps, |g_a| + |g_n|)} \tag{2.1}$$

Where $g_a$ and $g_n$ are analytical and numerical gradients respectively. Method 2 checks if absolute error is small:

$$error = |g_a - g_n| < tol) \tag{2.2}$$

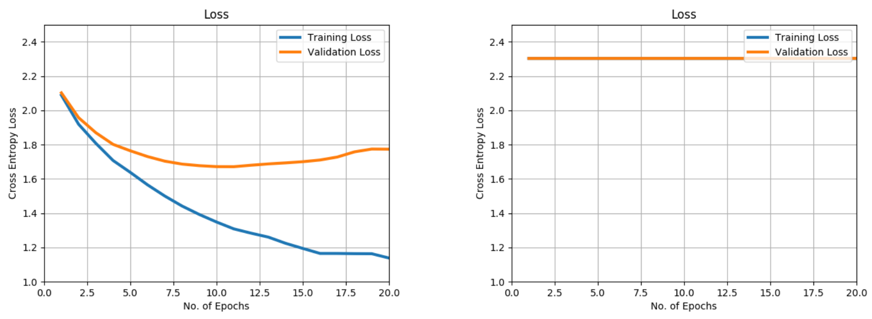
Where 'tol' is tolerance on error. The fast version gave really high values of error for small 'h' (step size). Hence for comparison, the slow version of numerical computation was chosen. 100 images were used for gradient comparison. To reduce computation time, first 300 dimensions of these images were considered. Table 1 gives error values for both methods. Note that for both Methods, $g_a$ as well as $g_n$ would be matrices. Hence, average value of all elements of matrix were calculated for comparison. The gradient computations were bug free.

Table 1: Gradient Computation error for h = 1$e^{-}$6.

| | Avg(Wt) | Avg(b) |
|---|---|---|
| Method 1 | $6.96 \times 10^{-6}$ | $5.35 \times 10^{-11}$ |
| Method 2 | $6.96 \times 10^{-6}$ | $3.47 \times 10^{-10}$ |

## 3. Batch Normalization and Training

To compare the effect of batch normalization on training, the evolution of loss function was observed for a 3-layer network with and without batch normalization.



(a) With batch normalization (b) Without batch normalization

Figure 1: Loss function for a 3-layer network, *'hiddenLayerSizeList'* = [50,30]. $\eta = 0.001$, $\lambda = 0$, nEpoch = 20

Without batch normalization, the hidden unit activations in deeper layers may saturate very quickly, which makes the learning almost impossible. On the other hand, batch normalization reduces the covariance of hidden layer activations, thereby avoiding this saturation.

## 4. Hyperparameter Search: Coarse to Fine

For a coarse search, a broad range was chosen for eta and lambda. The 3-layered network was trained for 15 epochs and 75 random pairings of eta and lambda. The training and validation set were chosen as batch-1 and batch-2 of CIFAR 10 dataset. The 'test batch' of CIFAR-10 was chosen as test set. The top-3 accuracy results are shown in Table along with ranges of eta and lambda chosen. 2.

| $\eta \in [10^{-3}, 10^{-1}]$ | $\lambda \in [10^{-9}, 10^{-2}]$ | **Accuracy (%)** |
|---|---|---|
| $1.751 \times 10^{-3}$ | $1.728 \times 10^{-9}$ | 46.80 |
| $1.805 \times 10^{-3}$ | $9.147 \times 10^{-3}$ | 45.25 |
| $1.122 \times 10^{-3}$ | $8.307 \times 10^{-6}$ | 42.64 |

Table 2: Best Results for coarse hyperparameter search. nEpoch = 15, nIter = 75.

Another search was performed for narrower range of eta and lambda. But this gave poorer results compared to first test. Lastly, a third exhaustive search was conducted with same broad range of eta and lambda as first test, but with larger random pairings (100). The results obtained didn't improve those obtained in Table 2 significantly.

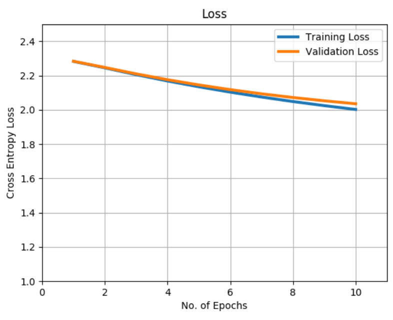
Finally, to test the best accuracy of 3-layer network, mini batch gradient descent was performed on 49000 training images and 1000 validation images (batch-1 to 5 of Cifar-10 dataset). The best performing values of eta and lambda were chosen (eta = 1.751e-3, lambda = 1.728e-9). Accuracy was computed on the test set (10000 images). The resulting accuracy obtained after 15 epochs was **50.14%**.

## 5. Batch Normalization and speed of training

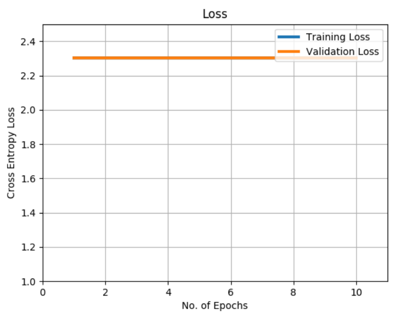
The training and validation loss for 2-layer network (50 neurons) was compared with and without batch normalization. Three values of learning rates were chosen (lambda was chosen as 0):

- Small: 0.0001
- Medium: 0.001
- High: 0.1

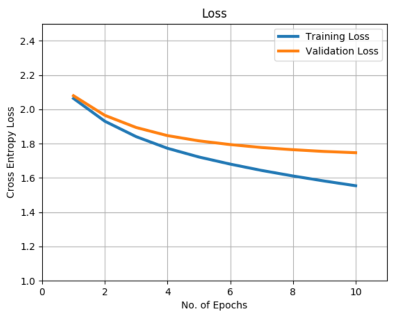
The results are shown in Figure 2. It was observed that batch normalization speeds up the training (reduction in loss was obtained for fewer epochs). The reason could be that batch normalization reduces the internal covariance shift of hidden layer activations. Hence if one of the neurons has learnt something wrong and wants to correct itself, it has to take fewer steps to reach the correct activation. It was also seen that BN reduces over-fitting, indicating that it leads to more stable training.



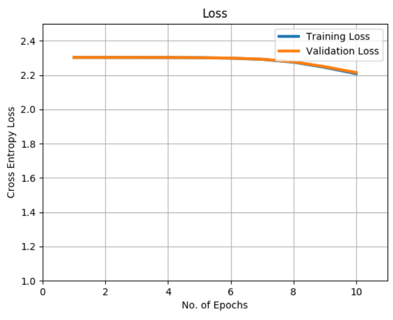
(a) With BN, $\eta = 0.0001$



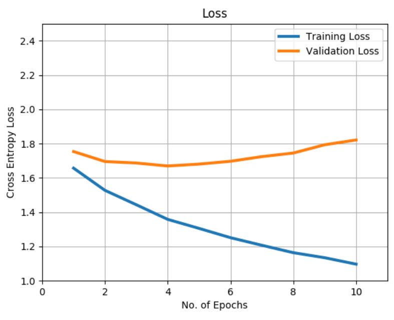
(b) Without BN, $\eta = 0.0001$



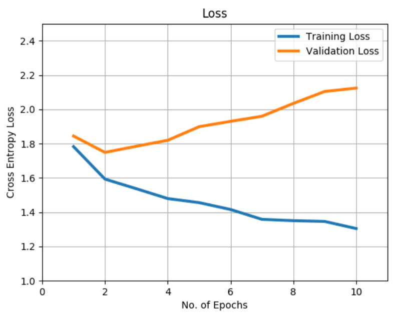
(c) With BN, $\eta = 0.001$



(d) Without BN, $\eta = 0.001$



(e) With BN, $\eta = 0.1$



(f) Without BN, $\eta = 0.1$

Figure 2: Total Loss on Training and Validation Data v/s No. of epoch