DD2424 - Deep Learning in Data Science
Assignment 2 Report

Ajinkya Khoche
khoche@kth.se

September 4, 2018

1. Objective

Continuing from Assignment 1, in this lab, a two layer Neural Network was created and tested on Cifar-10 dataset. Section 2. gives evidence for matching analytical and numerical gradient computations. Section 3. comments on improvements in speed before and after momentum. Section 4. and Section 5. talk about coarse and fine parameter search respectively and present hyperparameters for three best network configurations. Lastly Section 6. presents performance for given network settings and best-found hyperparameter configuration.

NOTE: Unless otherwise specified, the batch size was taken to be 100 and momentum constant ρ was taken to be 0.9.

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. The first 1000 dimensions of one image were used for gradient comparison. Table 1 gives error values for both methods. Note that for both Methods, g_a as well as g_n would be matrices. Hence, value of first element was taken for comparison. The gradient computations were bug free.

3. Momentum

Momentum term is added to Gradient Descent to speed up the training. Concretely, the update step is modified as follows:

Table 1: Gradient Computation error for h = 1e^-6 and first 1000 dimensions of one image.

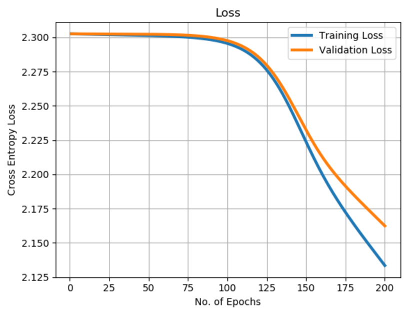
	Layer 1		Layer 2	
	W	b	W	b
Method 1	4.69×10^{-8}	3.26×10^{-7}	1.65×10^{-7}	1.08×10^{-7}
Method 2	3.62×10^{-11}	2.36×10^{-10}	1.27×10^{-10}	8.37×10^{-11}

$$v_t = \rho v_{t-1} + \eta \frac{\partial J}{\partial \theta} \tag{3.3}$$

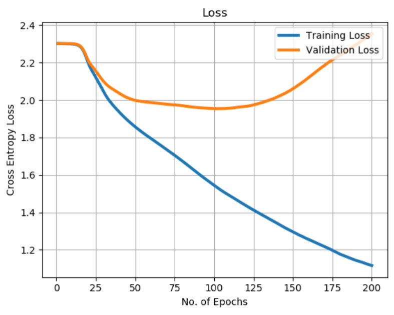
$$\theta_t = \theta_{t-1} - v_t \tag{3.4}$$

Where θ_i are Weights or biases. Unless otherwise specified, the value of momentum term (ρ) is assumed to be 0.9. One way to visualize effect of momentum is that it includes all previous gradients and reduces the importance of instantaneous gradient. In this way, a sudden hike in gradient is ignored and descent towards minima is smoothened.

To check the influence of momentum, mini batch gradient descent was computed with and without momentum. Figure 1 shows speed improvement with momentum.



(a) $\rho = 0$



(b) $\rho = 0.9$

Figure 1: Mini batch gradient descent with and without momentum. Parameters: 1000 images on training and validation set, $\eta = 0.005$, nEpoch = 200

4. Hyperparameter Search: Coarse

The following list of values were chosen for η and λ as initial step:

$$\eta \; \epsilon \; [0.0001, 0.0005, 0.001, 0.005, 0.008, 0.01] \tag{4.5}$$

$$\lambda \; \epsilon \; [0.001, 0.005, 0.01, 0.05, 0.1, 0.5] \tag{4.6}$$

First, a search for η was performed. λ was chosen to be 0. Gradient Descent was computed for first 1000 images from training set with 200 epochs. First 1000 images from validation set were used to compute accuracy. Momentum term was set to 0.9. The results of gradient descent are shown in Figure 2. From figure, an η value of 0.005 was found to give a good learning rate. Hence η was fixed to this value and a coarse search for λ was performed. The results are shown in Figure 3.

The above test provided an idea of a feasible range of values of η and λ. For this range, 50 random pairings were chosen and mini-batch GD was run on a new training set (49000 images from batch 1 to batch 5) and validation set (last 1000 images of batch 5). Number of epochs was chosen to be 15. The accuracy was computed on validation set. The hyperparameter settings for three best performing networks, along with chosen range of η and λ is listed in Table 2.

$\eta \, \epsilon \, [10^{-3}, 10^{-1}]$	$\lambda \, \epsilon \, [10^{-9}, 10^{-2}]$	**Accuracy (%)**
8.53×10^{-3}	2.27×10^{-5}	42.1
1.19×10^{-2}	4.04×10^{-7}	42.1
1.83×10^{-2}	8.99×10^{-6}	41.9

Table 2: Best Results for coarse hyperparameter search. nEpoch = 15, $\rho = 0.9$.

5. Hyperparameter Search: Fine

Based on coarse search, the following range was chosen for η and λ:

$$\eta \, \epsilon \, [10^{-3}, 10^{-2}] \tag{5.7}$$

$$\lambda \, \epsilon \, [10^{-9}, 10^{-4}] \tag{5.8}$$

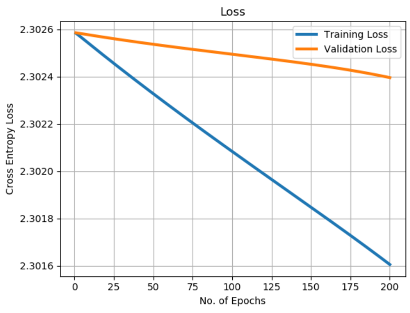
For above ranges, random pairings of η and λ were chosen. For every pair, mini batch Gradient Descent algorithm was run for chosen number of epochs. The training was done for entire training set and accuracy was computed on the validation set. The best results from 'Test 1' are displayed in Table 3 along with chosen parameter settings.

$\eta \, \epsilon \, [10^{-3}, 10^{-2}]$	$\lambda \, \epsilon \, [10^{-9}, 10^{-4}]$	**Accuracy (%)**
9.23×10^{-3}	8.09×10^{-8}	43.0
1.68×10^{-3}	2.34×10^{-8}	42.5
3.28×10^{-3}	1.58×10^{-5}	42.4

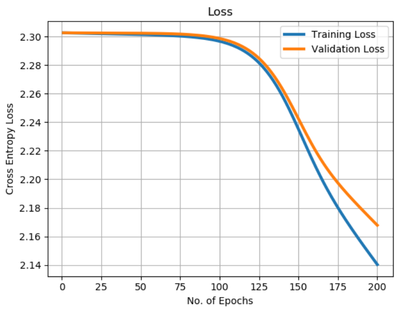
Table 3: Results of Fine Hyperparameter search Test 1. Number of pairings tried: 50, Number of Epochs: 10

6. Result with best hyperparameter setting

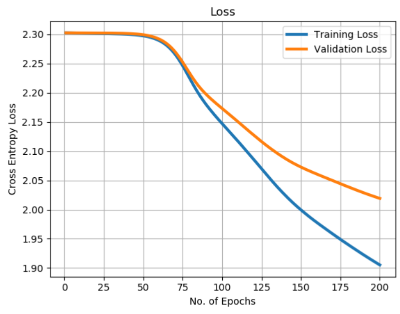
Mini-batch gradient descent was computed for best hyperparameter settings found in section 5. (eta = 9.23e-3, lambda =8.09e-8). Again, the training set consisted of 49000 images and the validation set consisted of 1000 images from Cifar-10 dataset. Training was carried out for longer duration (30 epochs) and the resulting weights tested on the test set (10000 images). The resulting accuracy obtained was **50.18%**. The training and validation loss after each epoch of training is shown in Figure 5



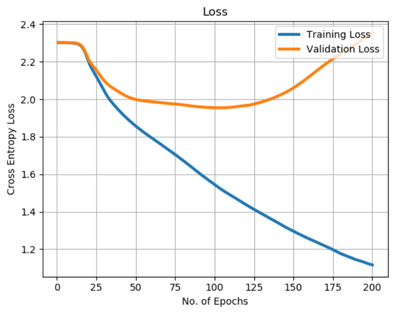
(a) $\eta = 0.0001$



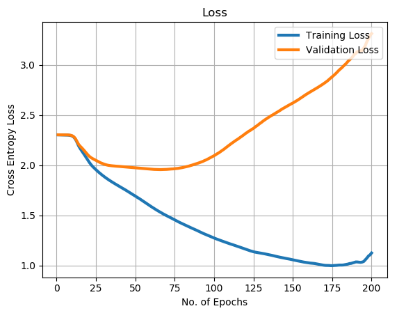
(b) $\eta = 0.0005$



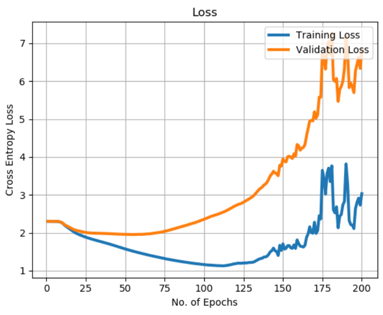
(c) $\eta = 0.001$



(d) $\eta = 0.005$

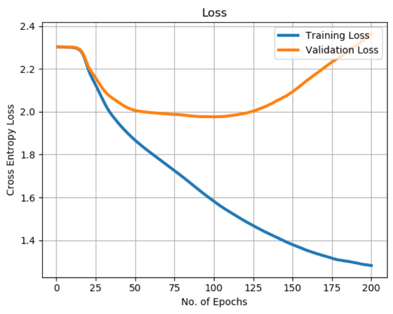


(e) $\eta = 0.008$

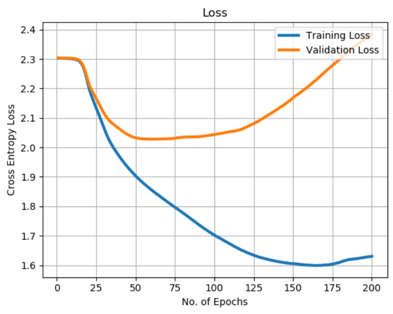


(f) $\eta = 0.01$

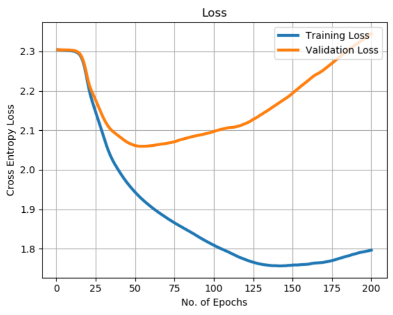
Figure 2: Total Loss on Training and Validation Data v/s No. of epoch



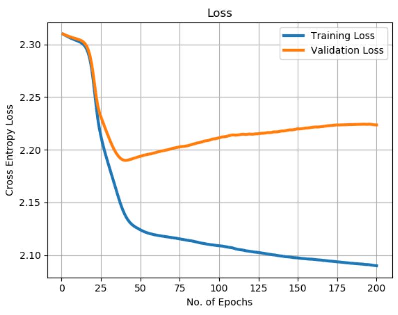
(a) $\lambda = 0.001$



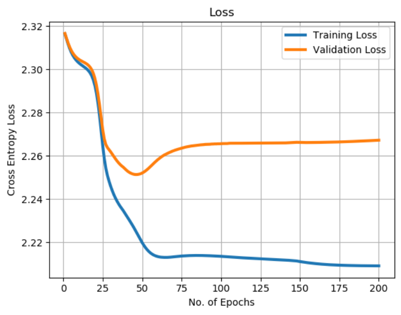
(b) $\lambda = 0.005$



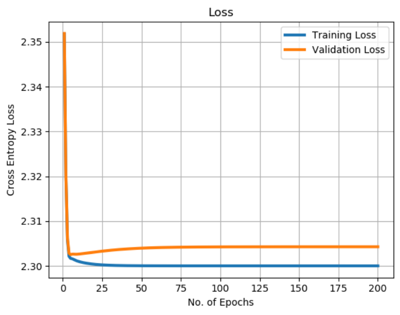
(c) $\lambda = 0.01$



(d) $\lambda = 0.05$



(e) $\lambda = 0.1$



(f) $\lambda = 0.5$

Figure 3: Total Loss on Training and Validation Data v/s No. of epoch

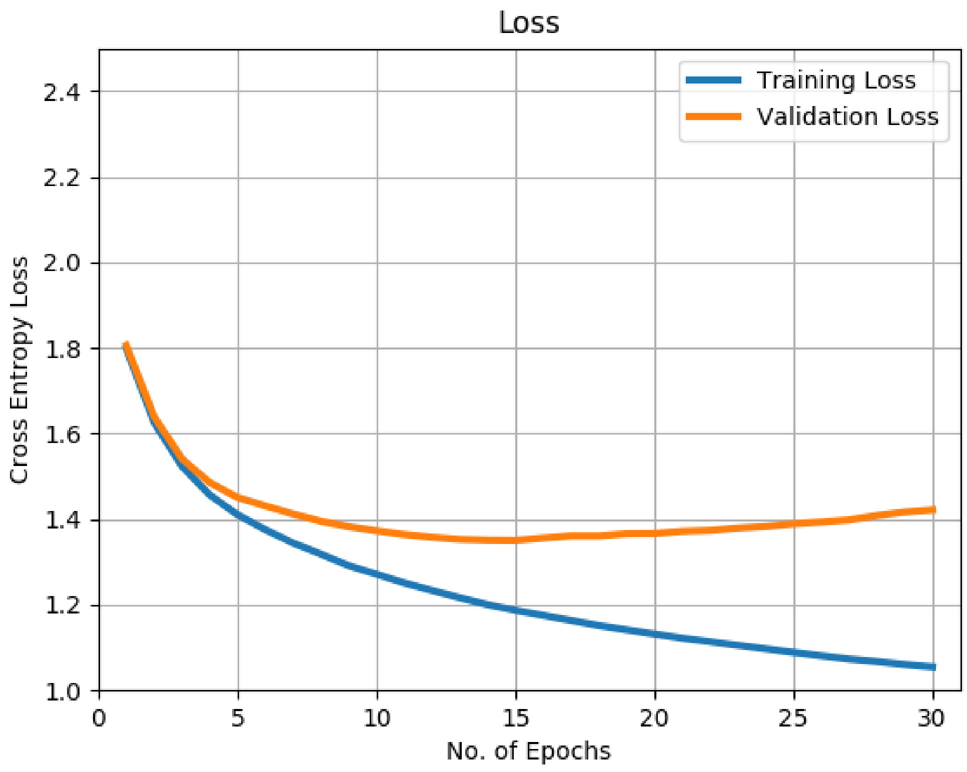


Figure 4: $\rho = 0$

Figure 5: Mini batch gradient descent training and validation loss, $\eta = 9.23e-3$, $\lambda = 8.09e-8$, nEpoch = 30