**­Assignment 3**

Diego González Morín

[diegogm@kth.se](mailto:diegogm@kth.se)

1. Objective

The main objective of this assignment is to train and test a k-layer network with multiple outputs for image classification. The training will be done using the Mini-Batch Gradient descent algorithm, with Batch Normalization, computing a cost function minimization. This cost function computes the cross-entropy loss of the classifier, applied to the labelled training data, and a L2 regularization term on the weight matrix. The dataset we will use for training, validation and test is the CIFAR-10 dataset.

1. Method

The assignment is divided in three parts:

1. In this first part the different functions that were written for the second lab will be adapted for this k-layer network.
2. Implementation of the Batch Normalization algorithm, necessary to ensure a correct training of a >2 layers network. Important to check the correct performance of the analytic gradients computation.
3. In the second part, different networks are trained using Batch Normalization or not, to compare performances and draw conclusions.
4. In this last section, the coarse to fine random search of the main parameters of the network (eta and lambda) is done using Batch Normalization. Also, a 3 layer network is trained with all the data available using the best combination of parameters found.
   1. Function implementation and verification

The first step then was to adapt all the functions implemented in the second lab to work on a two layer network. This functions are:

* function [X, Y, y] = LoadBatch(filename) – takes a input filename of a dataset file, loads the file, extract the pixels data in a matrix with dimensions d(dimension of each image)xN(number of images), the one-hot representation of the images with dimension K(number of labels)xN(number of images) and the vector y that contains the labels for every image N.
* function P = EvaluateClassifier(X, W, b) – Computes the probability of each label for each image, using the weight matrix, the vector b and the input data. The size of P is K(number of labels)xn(number of pictures in the input data).
* function J = ComputeCost(X, Y, W, b, lambda) – Computes the cost/loss of the network depending on the weight matrix (W) the vector b, the regularization term (lambda), the one-hot representation(Y) and the input data. This function is based on the equations available in the lecture notes. The output is a scalar.
* function acc = ComputeAccuracy(X, y, W, b) – Computes the proportion of images of the test data that were correctly classified by the net, using the input data(X), the labels for image in the set(y), the weight matrix(W) and the vector b.
* function [grad\_W, grad\_b] = ComputeGradients(X, Y, P, W, lambda) – Computes the gradients for the Weight matrix W and the vector b. The implementation of this method is based on the operations available in the lecture notes. We have that grad\_W has size K(labels)xd(dimension of each image) and grad\_b has dimension K(labels).
* function [Wstar, bstar, val\_loss, train\_loss] = MiniBatchGD(X, Y, X\_val, Y\_val, GDparams, W, b, lambda, display) – Trains the input Weights and b vector using the Mini Batch Gradient descent algorithm. To do so, it computes the probability matrix, then the gradients, and the W and b are then updated accordingly to this gradients and the learning rate eta. It did not use the validation data to avoid over fitting, however the addition of this parameter would improve the performance of the network.

To check the performance of these functions, the output for the second lab functions and these rewritten function was compared using the same 1 hidden layer network and parameters in both cases. For all the functions, zero error between outputs was obtained, and it can be concluded that they were correctly modified.

* 1. Batch Normalization and Gradient Check.

Training a network with 3 or more layers is not possible using just random initialization. Therefore, Batch Normalization must be added to overcome this limitation. Once the Batch Normalization algorithm was added to the code, the most important step was to check the performance of the function that computes the Backward pass (gradients). To do so, the first step was to compare the output of this function with the outputs of a pre-written function that calculates the gradients numerically. To do this, a small subset of the available data was used. After some experimentations, the error between both gradients was small enough (generally smaller than 1E-4).

However, an extra sanity check was done. As it was recommended in the lab notes, we double checked its performance by training the network on a small amount of training data and check if we can overfit to the training data and get a very low loss using a reasonable value of eta. In Figure 2 we can observe how the network overfits the training data, as the training loss is very low while the validation loss increases considerably.

[][]

The network overfits completely to the training data as the training loss is almost zero and the validation loss increases considerably. It can be assumed that, after satisfying both requirements (numerical compare and network overfitting), the gradients are well implemented.

* 1. Batch Normalization Experimentation

It was stated before that training a >2 layer network can become an impossible task if Batch Normalization is not implemented. Besides, learning is faster when Batch Normalization is applied. In this section, the necessary experimentation is done to give proof of these two statements.

* + 1. Loss evolution for a 3 layer network

In this first experiment, the loss evolution during training with and without Batch Normalization of a 3 layer network will be measured and compared. This loss comparison is shown in the figure [].

* + 1. Training and Validation loss evolution for a 2 layer network

The experiment is repeated, but using a 2 layer network instead. 3 different values of etas will be used ( high, medium and low ones) to perform the comparison between the training with and without Batch Normalization. 10 epochs were used in all the experiments. The results are shown in figure [].

* 1. Parameters tuning for 3 layer network
     1. Coarse to fine random search of eta values range

The first step in the process of parameters tuning is to find a reasonable range of values for the learning rate. To do so, we are going to set the regularization term to a very small value ( .000001). After the 3 layer network is initialized randomly using Xavier’s initialization, the next step is to perform a quick search by hand to find the rough bounds for reasonably values of the learning rate. After several experiments, the training loss is shown in Figure []. This training losses were calculated with the data in the first batch ( data\_batch\_1.mat) and only 2 epochs, as this number of epochs was enough to estimate a coarse range for the etas values.

In figure 4 we can observe that for eta = [], the evolution of the loss is considerably slower compared to the rest of loss. Also, from eta >= [] we are getting values of the training loss that stop decreasing after 2 epochs. This is way, the chosen range of feasible values of eta were [].

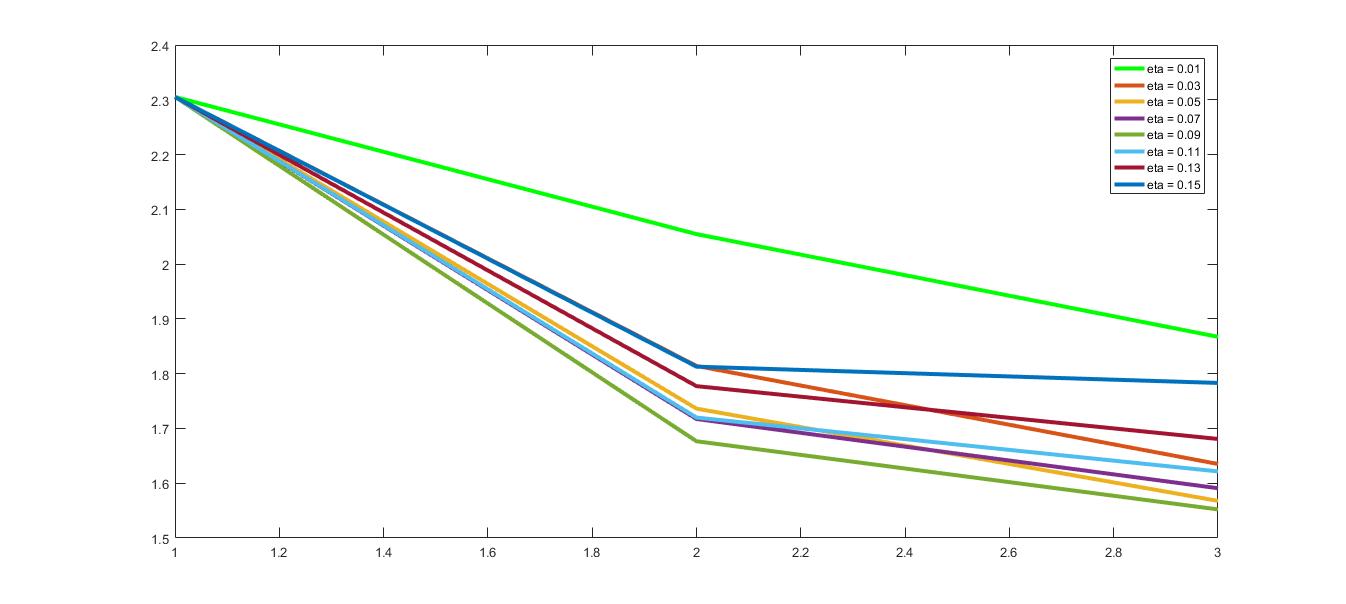


Figure 4: Training loss after training a network for 2 epochs for different values of eta and with lambda fixed to 0.000001.

* + 1. Coarse to fine random search to set lambda and eta

After the range of reasonable values of eta is estimated, the next step is to fine tune both lambda and eta parameters. To do this, a random search will be performed. After the network is randomly initialized using Xavier initialization, it is trained several times for different parameters of eta and lambda. The etas will be chosen randomly within the estimated range of feasible-rates of eta in tandem with a search over a very broad range of values of lambda. To make sure that all the range of values was included when randomly pick the etas, the estimated range of feasible-rates of eta was made wider. Therefore, the eta used in this first random search was eta = [] instead of eta = [] calculated. As we are doing a random tandem search, each sample of eta and lambda will be randomly picked using the following expression:

e = e\_min + (e\_max – e\_min)\*rand(1,1);

To estimate the tandem of optimal parameters, 60 random picked pairs were used, to train the randomly initialized network (with the same seed always), during only one epoch. Only one epoch was used as this is still a coarse search, and this was a quick way to make the ranges of eta and lambda narrower. All the scores both of the validation loss and the accuracy of the network on the validation data were saved. The pair of parameters the gave the better results ( higher accuracies or lower losses) were stored. Then, the higher and lower boundaries of eta and lambda that gave the first 15 better results were computed and stored. This range of optimal parameters was eta = [] , lambda = []. This boundary values of the parameters were used to repeat the process with a narrower range, to try to get closer to the optimal parameters. As previously, we are using a slightly wider range, to make sure that all the possible optimal parameters are chosen during the random search. We used: eta [] , lambda = [] This second random search was done with 60 pairs of parameters during 3 epochs. Only two epoch more were used as the main improvements in the accuracy when training the network with only one batch of data occur during the first 3 epochs. In this case, the range calculated was eta = [] , lambda = [], being the optimal pairs of parameters the ones shown in table 1. To compute this final range, only the best 8 combinations were chosen. The best pair of parameters chosen was the one with better performance during the random search: eta = [] and lambda = [].

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Eta\_range | Lambda\_range | Epochs | Number of pairs trained/considered | Eta estimated | Lambda Estimated |
| 0.02 – 0.1 | 0.0005 – 0.1 | 1 | 60/15 | 0.00538-0.141 | 0.000129- 0.0087 |
| 0.00053 – 0.141 | 0.000013 – 0.0087 | 3 | 60/8 | 0.0319-0.0858 | 0.000383-0.0061 |

Table 1: Ranges for Eta and Lambda

1. Training

After the best tandem of parameters was calculated, a new randomly initialized network was trained using this parameters and all the training data available. The dataset was then organized as follows:

* Trainning set : data\_batch\_1.mat, data\_batch\_3.mat, data\_batch\_4.mat, first 9000 images of data\_batch\_2.mat.
* Validation set: last 1000 images of data\_batch\_2.mat.
* Test set: test\_batch.mat

The best performances were obtained with the pair of paramters eta = [] and lambda = [], with rho, as before, set to 0.9 and the eta decay to 0.95. The network was trained for 20 epochs, obtaining a final accuracy on the test set of []. In figure 6 we can see the training and validation loss during training.

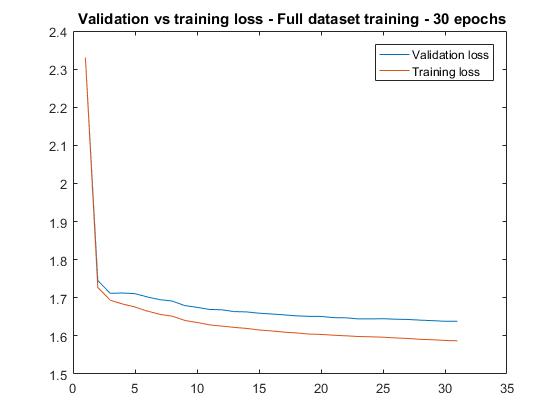


Figure 6: Validation and training loss during the training over 20 epochs, for eta = [] and lambda [] using all the available batches for training.