Assignment 1

Diego González Morín

diegogm@kth.se

1. Objective

The main objective of this assignment is to train and test a single layer network with multiple outputs for image classification. The training will be done using the Mini-Batch Gradient descent algorithm 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.

2. Method

The assignment is divided in two parts: a first part in which different functions have to be implemented and a second part that, using this written functions, trains a network using the Mini-Batch Gradient Descent algorithm.

2.1. Function implementation and verification

The first step then was to build all the functions required for the later implementation of the training algorithm. 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). This is the key function, as the Gradient descent is based in this function and its debugging can be hard. To successfully debug this function, the recommended option was to compare the performance of the function (analytical gradient calculation) with the results obtained with

the numerical gradient computation. If the difference was small enough (< 1e-6) we could consider that the function worked well. .

- `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.

3. Results and conclusions

Once the functions were implemented, the network was trained using different combination of eta, and regularization term (lambda)., being the number of epochs and size of the mini batches fixed to 40 epochs and 100 respectively. The results for each proposed combination of parameters were:

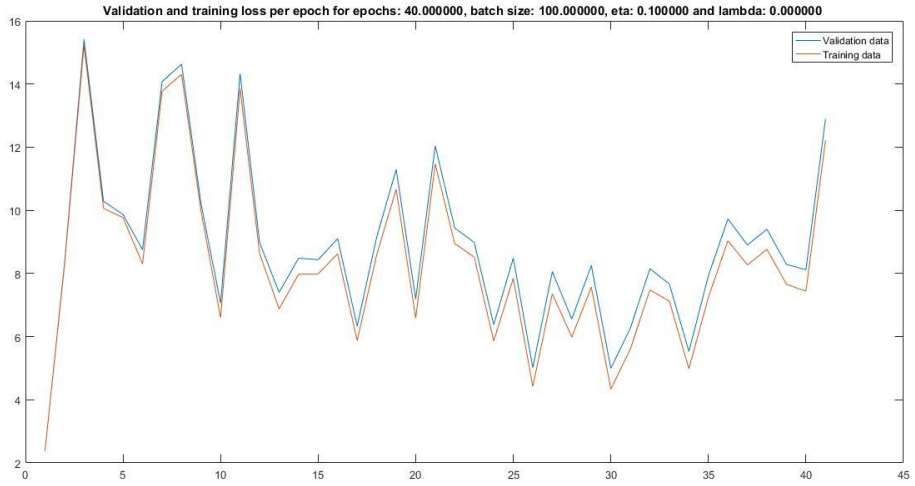

Figure 1: Validation and training loss for eta = 0.1, lambda = 0

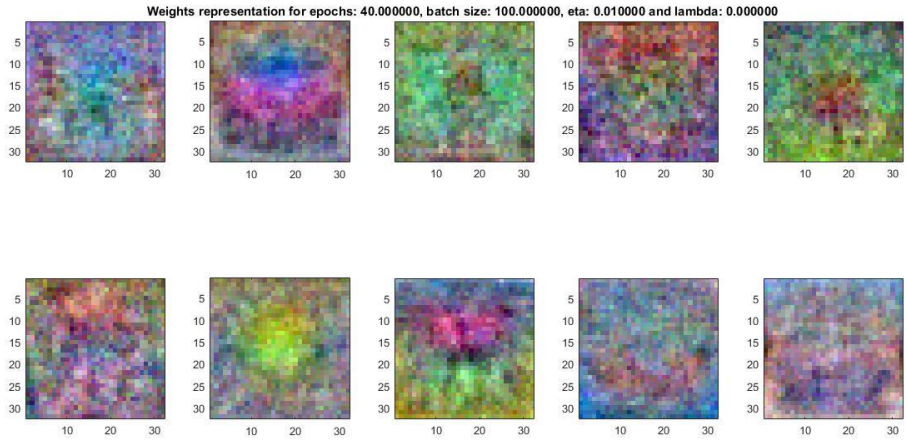

Figure 2: Weight matrices representation for eta = 0.1, lambda = 0

Is easy to observe that the evolution of the loss in the figure 1 is really unstable. If we observe the other figures, we can conclude that the only combination of parameters that have this unstable

tendency of the loss is the first one. This is due to the higher learning rate that makes very difficult for GD algorithm to converge a local minima. Due to this phenomena, the accuracy is very low – accuracy = 17.4 %.

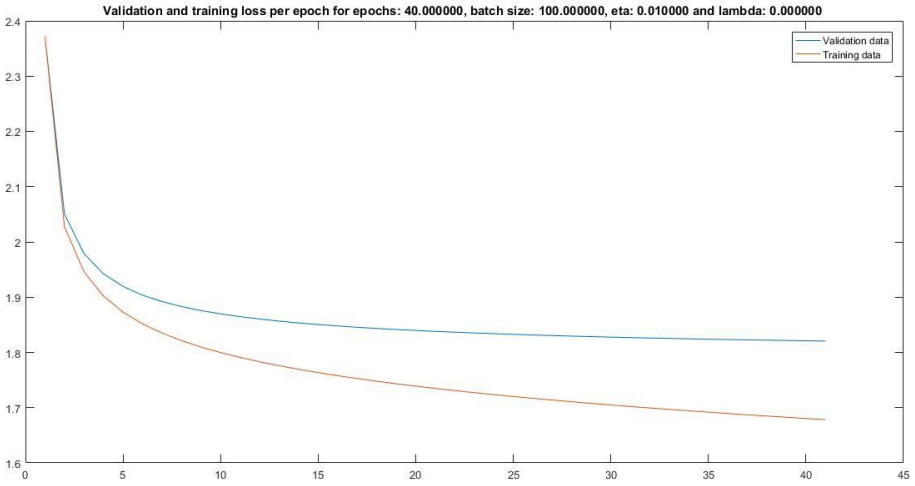

Figure 3: Validation and training loss for eta = 0.01, lambda = 0

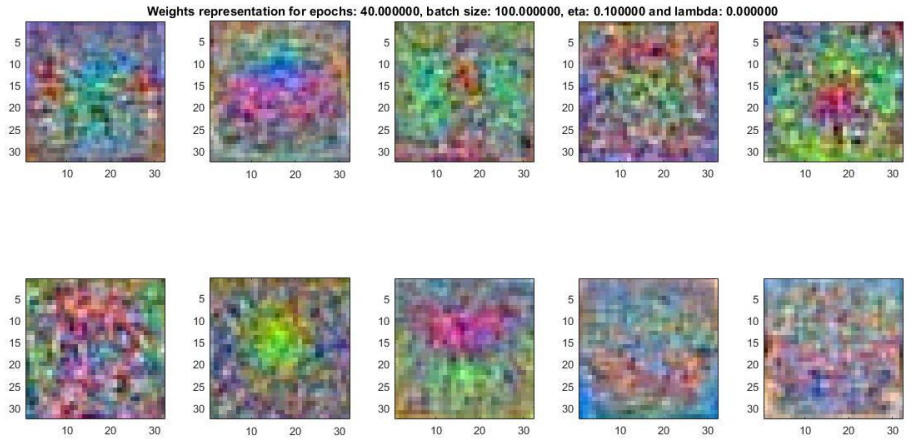

Figure 4: Weight matrices representation for eta = 0.01, lambda = 0

If we fixed all the parameters to the same value but we make the learning rate ten times bigger, we get the validation/training loss represented in the figure 3. We can see that this curves are now stable, and the loss decrease gets smaller as it gets closer to a local minima. The accuracy has increased considerably, being this combination of parameters the best in terms of accuracy – accuracy = 36.65%.

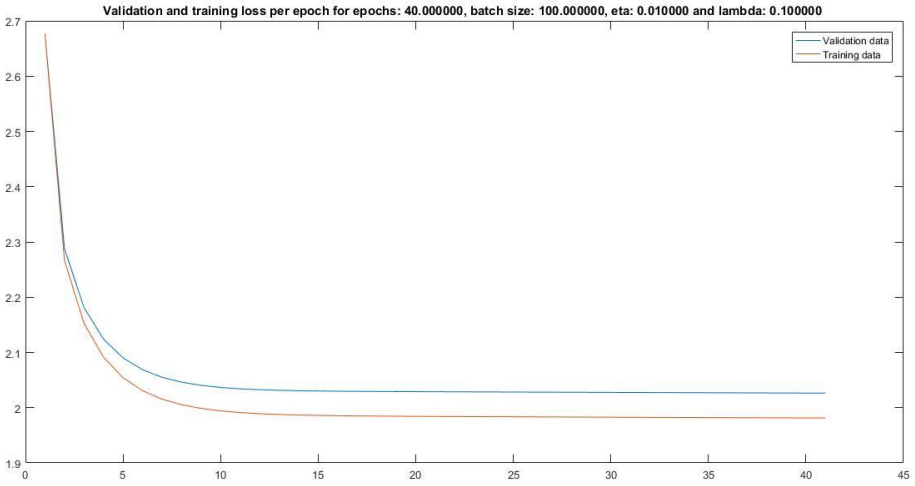

Figure 5: Validation and training loss for eta = 0.01, lambda = 0.1

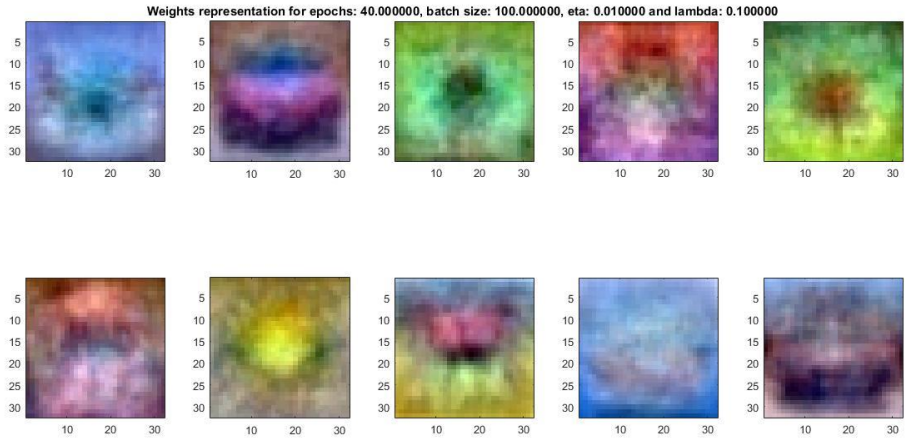

Figure 6: Weight matrices representation for eta = 0.01, lambda = 0.1

In this case, we use the same parameters as in the previous example, but changing the regularization term (lambda). Now we can observe that validation and training loss are closer to each other and the local minima is reached much faster. Reaching the local minima faster does not mean an improvement in the performance as we can observe that the accuracy has decreased. The closer the 2 curves are, the less probability of over fitting we have. As a consequence, we can conclude that the decrease in the accuracy corresponds to an under fitted model . Accuracy = 33.37%

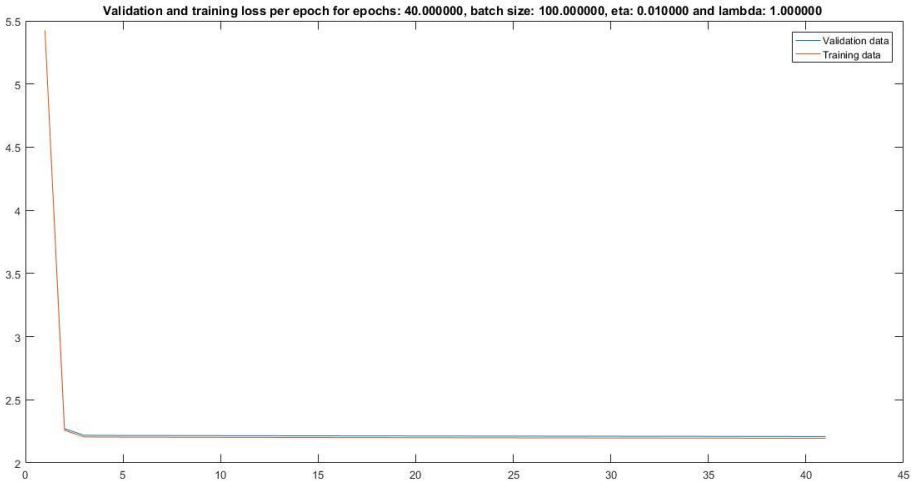

Figure 1: Validation and training loss for eta = 0.01, lambda = 0.1

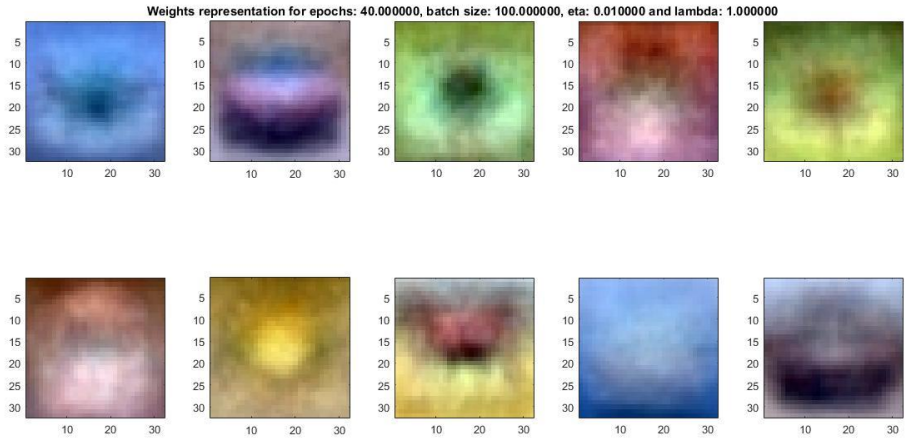

Figure 8: Weight matrices representation for eta = 0.01, lambda = 1

Increasing the regularization term even more gives the loss represented by the figure 7. The under fitting phenomena is even stronger now (the validation and training loss curves are even closer to each other), decreasing the accuracy considerably. – Accuracy = 21.92%.

The accuracies for each combination of parameters are gather in the following table:

Num. of epochs	Batch size	Eta	Lambda	Accuracy
40	100	0.1	0	17.4
40	100	0.01	0	36.65%
40	100	0.01	0.1	33.37%
40	100	0.01	1	21.92%.

Table 1: Accuracies in the networks trained with different combination of parameters