DD2424 Deep Learning in Data Science

Assignment 1

Ali Shibli

April 2021

1 Introduction

In this assignment, we build a 1 layer neural network from scratch for classification task. The dataset being used is CIFAR-10 dataset, that consists of 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

2 Functions Implemented

2.1 LoadBatch

This function is simply to load the dataset from disk. There are 5 batches of files and 1 testing batch when we download from source. For this assignment, the first batch is used for training, second for validation, and third for testing (if required).

2.2 Normalize

This function basically pre-processes the dataset. It centeres the points (vectors) around 0 and divides by their standard deviation.

2.3 initialize_weights

This function initializes the weight matrix W and the biases vector b. We use a gaussian random initialization with mean=0 and std=0.01.

2.4 EvaluateClassifier

This function computes the predictions for a particular input data (batch) in terms of a probability distriubtion for each class label. In other words, given an input matrix (or vector) X, as well as the weights matrix W and biases vector b, it returns a vector containing the probability of each label from the target labels. Then we choose the label with highest probability as our class guess.

2.5 ComputeCost

This function computes the cost function (or loss) of our network. It derives from our predictions in the EvaluateClassifier function, and compares the results with the ground truth results. This cost is a combination of two costs. For the first part we are using cross entropy loss function. For the second term, it is the regularization loss with the corresponding λ parameters to help in generalization and regularization of the network.

2.6 ComputeAccuracy

This function computes the accuracy of the predictions. That is, the True Positives plus the True Negatives out of the total predicted examples.

2.7 ComputeGradients

This function is the main part (or heart) of the algorithm. This is where the parameters of the network W and b get updated at each iteration. In particular, we follow these steps to compute the gradients of these parameters:

1. Compute $G_{batch} = -(Y_{batch} - P_{batch})$
2. Compute gradient w.r.t. W as $\frac{dl}{dW} = \frac{1}{n} G_{batch} X_{batch}^T$
3. Compute gradient w.r.t. b as $\frac{1}{n} G_{batch} 1_n$

This calculation so far is analytical calculation. Another way is by numerical calculations, using functions pre-implemented. To verify our gradient computations, we furthermore experiment with the following two given functions:

- ComputeGradsNum: faster but less accurate results
- ComputeGradsNumSlow: slower but more accurate results

We use two metrics in the comparison between our computed gradients and the gradients obtained from the predifined functions. The first one is simply the average of the number of errors in the results, where the errors are simply the points which have a difference $> \epsilon = 10^{-6}$. The second metric is the relative error between our computed gradients and the obtained gradients. We take the also the points whose difference is larger than the threshold $\epsilon = 10^{-6}$.

As a first run, we start with no regularization term, that is, $\lambda = 0.0$. In addition, instead of taking the entire dataset, we take the first 20 dimensions of the first training example.

For this small set, both metrics result in zero errors. That is, all the differences are within the threshold value $\epsilon = 10^{-6}$.

Next, we add regularization term of $\lambda = 0.01$ and we increase the batch to take the first 5 samples instead of 1 with their entire dimensions.

Using the first metric we record zero errors between the computed gradients and the numerically computed gradients (for both functions). However, using the metric, we notice a small difference for the W matrix, where using the first function, we get 1877 errors (with error rate = 6.11%), whereas using the second (more accurate but slower) function, we report less errors, 12 errors (with error rate = 0.00039%).

2.8 MiniBatchGD

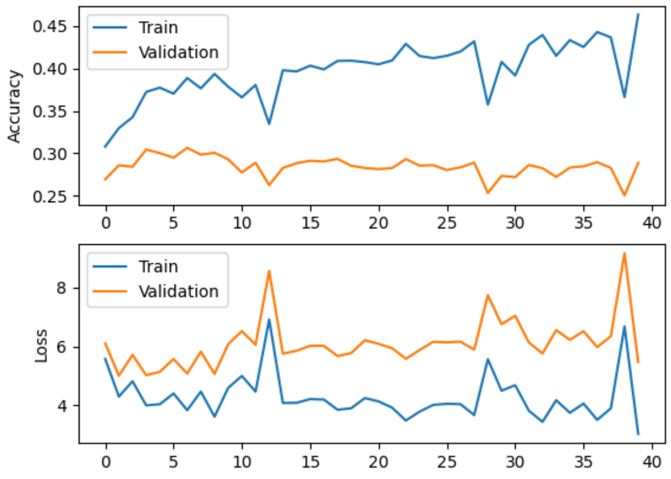
This function is the high level computation of the algorithm. We run over the different batch sizes of the data, computing the gradients at each iteration, and updating the weights and biases until the number of epochs is reached. We can visualize the learning curves for 4 different scenario presented in the table below:

Case	Lambda	Number of epochs	Batch size	Learning rate
1	0.0	40	100	0.1
2	0.0	40	100	0.001
3	0.1	40	100	0.001
4	1.0	40	100	0.001

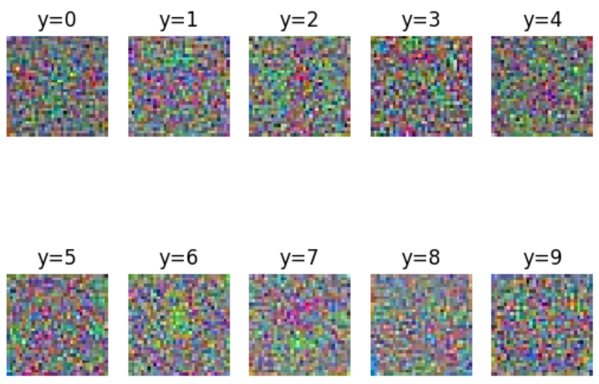
Next, we can visualize the results on each of these cases:

3 Results

3.1 Case 1



(a) Accuracy and loss graphs for train and validation sets

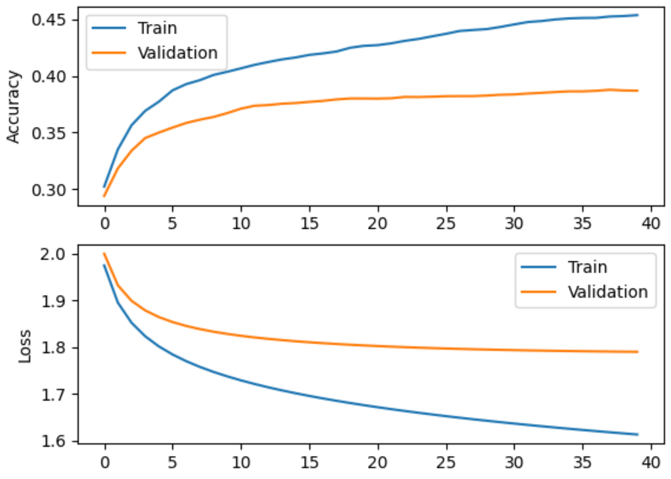


(b) Weight matrix visualization

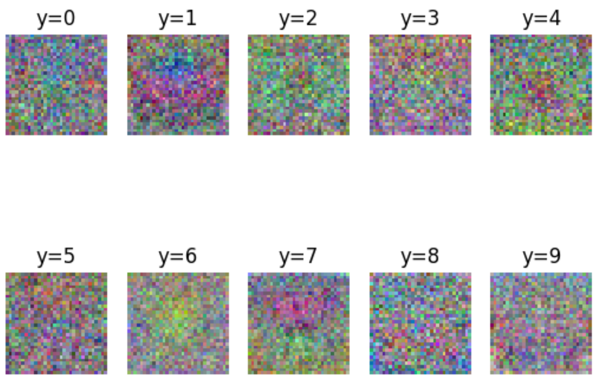
Figure 1: Case 1 results

Final test accuracy on testing set: 0.288
(We note of the unrealistic learning of this model due to improper parameters. We don't have regularization and we have high learning rate.)

3.2 Case 2



(a) Accuracy and loss graphs for train and validation sets

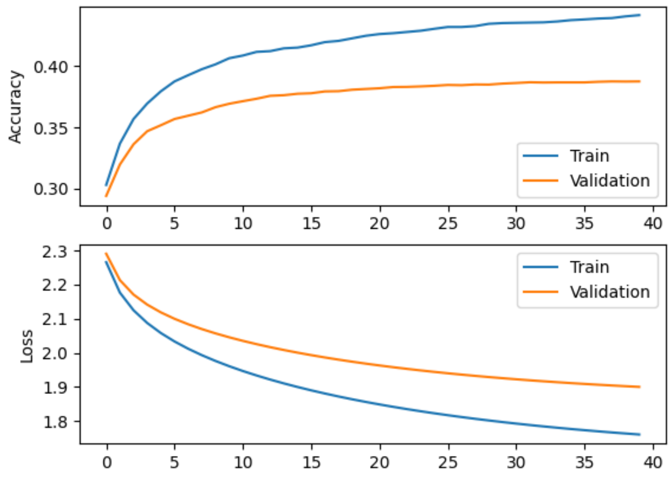


(b) Weight matrix visualization

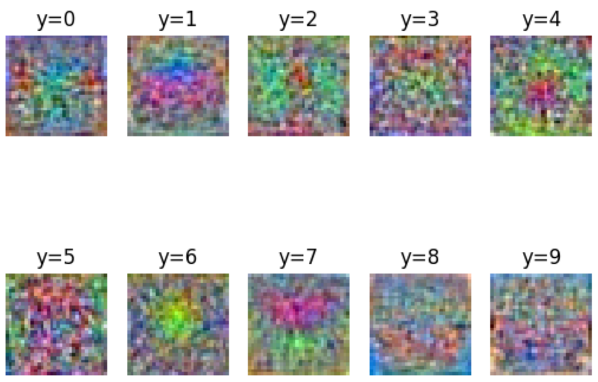
Figure 2: Case 2 results

Final accuracy on testing set: 0.3868

3.3 Case 3



(a) Accuracy and loss graphs for train and validation sets

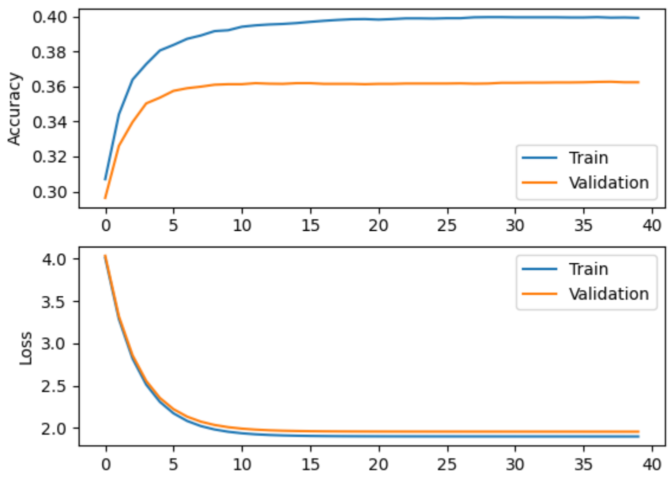


(b) Weight matrix visualization

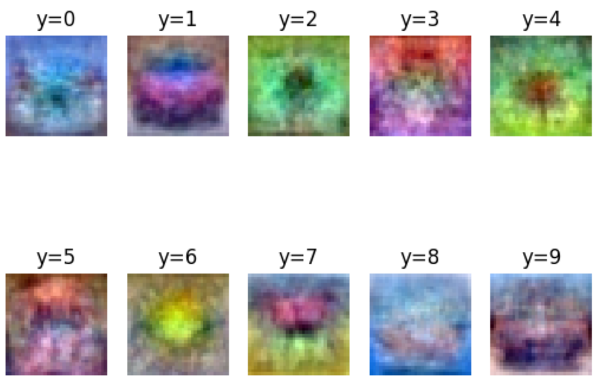
Figure 3: Case 3 results

Final accuracy on testing set: 0.3909

3.4 Case 4



(a) Accuracy and loss graphs for train and validation sets



(b) Weight matrix visualization

Figure 4: Case 4 results

Final accuracy on testing set: 0.3719

4 Conclusion

From the previous section, we notice the importance of the learning rate and the regularization parameters. As the regularization parameter Λ increases, we start getting better accuracies, however to some threshold. As we noticed that as λ increases from 0.1 to 1.0, the accuracy decreased. On the other hand, the learning rate μ also plays a fundamental role in learning. A high learning rate of 0.1 resulted in unstable learning of a spiky-shaped learning curve (non-continuous values). However, when the learning rate was dropped to 0.001, results started to become more continuous and stable.