KTH Royal Institute of Technology

Deep Learning in Datascience
DD2424

Assignment 1

Authors:
Ali Banaei Mobarak Abadi

April 4, 2022

1 Assignment Report

In this section, the results we can get by running the code are presented. For an explanation of the code and implementations, please see section 2. Note that for the rest of this report, the available training data is divided into two parts, validation with a size of 5000 samples and the rest as training.

1.1 Checking the gradients

After implementing the code for the forward path and calculating the gradient, calculated gradients were checked using the provided functions (for the final run, the slower version was used). To make the gradients big enough so that we can use the difference without a need for a division, the weight and bias matrices were initialized with a Gaussian distribution with a mean of 0 and a variance of 4. After calculating the gradients using the two methods, the mean, standard deviation, and maximum values for the absolute values of the difference of gradients were calculated by two methods, and the min, max and the SD of numerical gradients were printed. The results were as follows.

```
For abs of diff of gW: mean: 5.12e-09, std: 3.75e-09, max:2.11e-08, gradient min: -1.72,
  gradient max: 1.53, gradient std: 0.42
For abs of diff of gb: mean: 1.60e-09, std: 1.33e-09, max:4.21e-09, gradient min: -0.1,
  gradient max: 0.064, gradient std: 0.12
```

As we can see, the differences are vary small while the gradients are not. So, we can conclude we are calculating the gradients correctly.

1.2 Training and evaluation

After implementing the remaining functions, the model was trained and tested using the provided hyperparameters. Figure 1 depicts the loss, cost function, and accuracy of training and validation sets during training. Also, the results of the evaluation of the model on test data can be found in Table 1.

Table 1: Evaluation of the model with different values of λ on the test set.

λ	η	Loss	Cost	Accuracy
0	0.1	6.92	6.92	0.27
0	0.001	2.02	2.02	0.32
0.1	0.001	1.74	1.80	0.41
1	0.001	1.85	1.92	0.37

We can see that when we set 0.1 as the learning rate, the network reaches a not-so-bad point after the first iteration; however, it keeps overshooting the optimal values and does not converge to a good set of weights. In this setting, the network jumps around, and it does not have adequate performance in the evaluation. However, when we change the learning rate to 0.001, the network converges, and the final accuracy increases. It is worth mentioning that when we do not use regularization ($\lambda = 0$), the validation and training loss and accuracy have a considerable difference, and the network's performance is better on training data that it has seen. As expected, when we increase λ, this difference decreases, and the generalization performance of the network enhances. However, when we have $\lambda = 1$, the network's performance decreases. It may be because this value would enforce a strong constraint on our network, which structure is very simple. So, the network would not be able to optimize the weights to decrease the loss function effectively.

Also, the visualization of the W can be found in Figure 2. As we see, by increasing the λ, we have a smoother matrix. Also, when we do not use regularization, the network does not seem to capture any repetitive pattern in each class, while by using regularization, instead of memorizing the training set, the model learns these patterns.

2 Implementation Documentation

Here is a brief explanation of the implemented functions. For each one, a set of inputs, outputs, and a short description of it is provided.

unpickle: Loads a file saved using pickle. The input is the path to the file and returns the loaded onject.

load_data: The input is the path to the directory containing CIFAR-10 dataset. Then the function loads all the batches, concatenate them and perform z-scoring on all the data based on the mean and variance found

in training set. Then based on the input validation_n splits a validation set from the training data and return training, validation and test sets. Labels are represented by one-hot coding.

shuffle : Performs random shuffling of the input arrays. Note that arrays must have the same size in the second axis.

flip : Performs horizontal flipping of the samples in a dataset X.

softmax: Softmax activation function.

sigmoid: Sigmoid activation function

forward: forward path in the network.

accuracy: Accepts correct labels and predictions of a model and returns the accuracy of the model in predicting the labels.

loss: Cross entropy loss and cost. If we set `lamda=0` then the result is the loss and otherwise it is the cost.

mbce_loss: Multiple-MBCE loss.

compute_gradients: Calculates the gradient of the cost function with respect to W and b.

fit : Performs training on the dataset `X,Y`. This function can perform flipping agumentation (for bonus part) and training with M-BCE (bonus part). Function returns the final values of W and b with logs. Logs is a dictionart containing accuracy, lost and cost after each epoch for training and validation (if given) data during training.

evaluate: Accepts parameters of a model and evaluate it on the input dataset. Then accuracy, cost and loss values are returned using a dictionary.

gradient_checker: Used to check if our `calculate_gradients` function is working properly. This function initialize model parameter with big values and calculates the gradients of the cost function using both numeric and analytical solutions. Then some statistics are printed which help us evaluate the correctness of out implementations.

fit_and_plot: This function is used for the mandatory part of the assignment. It initialize a model, fit it on a given dataset, plot the logs, evaluate on test data and finaly plot the parameters W. Plots are saved in a subdirectory of the project.

mandatory: Performs tests needed for the first part of the assignment.

bonus_2_1_1: First method suggested in the bonus part to enhance the performance of the model. This function trains a network on a bigger training dataset and evaluate it on test data.

bonus_2_1_2: This function uses flipping as a agumentation method with probability of 0.5 during training. Then the trained model is evaluated on test data.

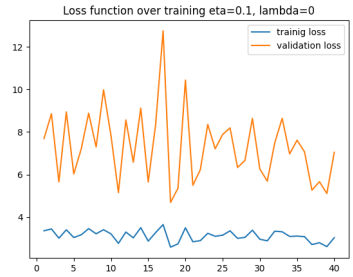
lr_scheduler: A scheduler for learning rate. Input is the epoch we are currently on and the function returns the learning rate we must use in this epoch. This function is used for learning rate decay during training.

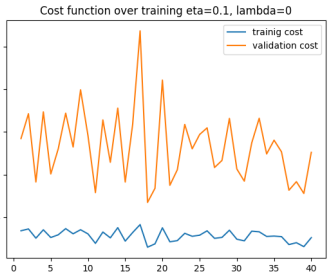
fit_with_scheduler: Instead of training a model with a fxed learning rate, uses the scheduler to set the learning rate at each epoch.

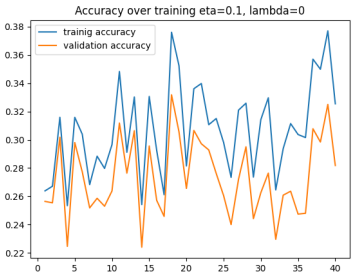
plot_correct_hist: This function plots the histogram of the probability of the correct classes and the ones correctly classified. This function is used in 2.2.

bonus_2_2: Trains two models with cross-entropy with softmax and M-BCE with sigmoid, plots the histograms and loss during training and evaluate them on test data. This function does everything for the second question of the bonus part.

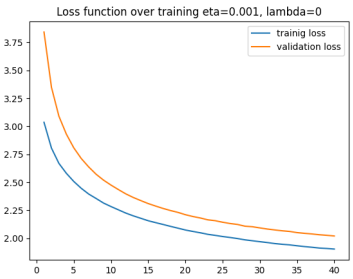
Please note that since these functions were originally implemented in different files, running the uploaded code will probably result in compile or runtime errors. If there you need to run the code, please contact me, and I can provide access to the Github repository containing the project for this assignment.

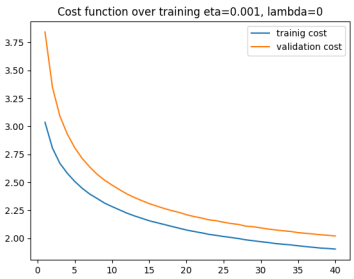


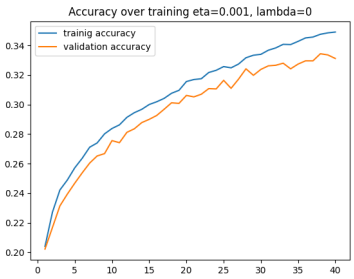




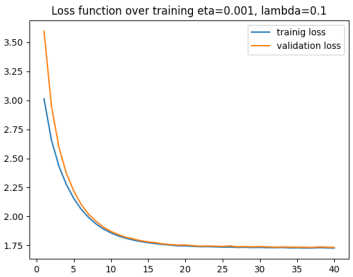
(a) $\lambda = 0,\ \eta = 0.1$

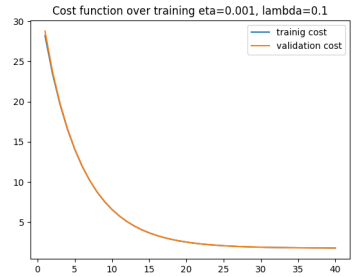


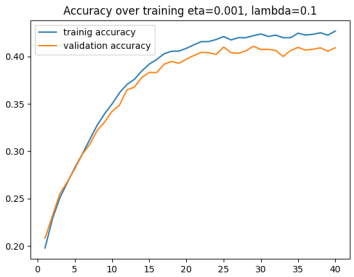




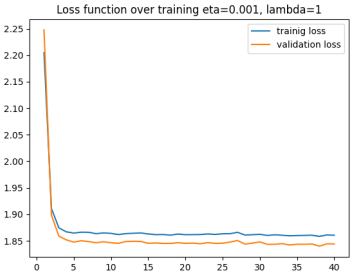
(b) $\lambda = 0,\ \eta = 0.001$

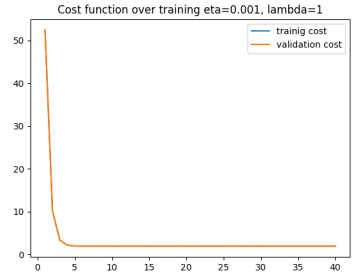


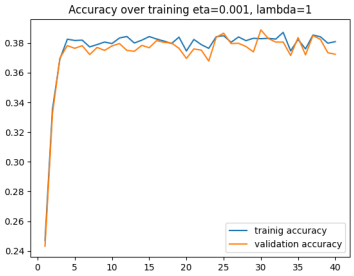




(c) $\lambda = 0.1,\ \eta = 0.001$

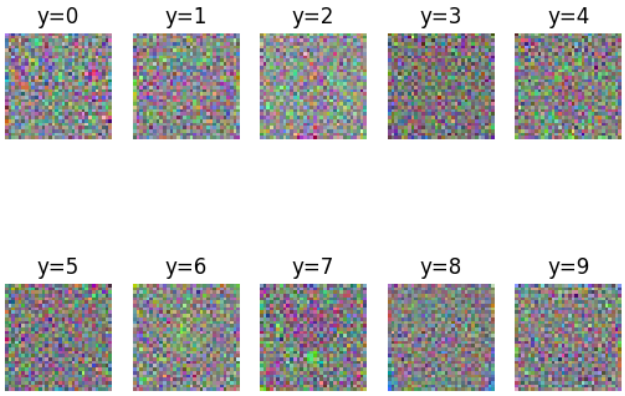




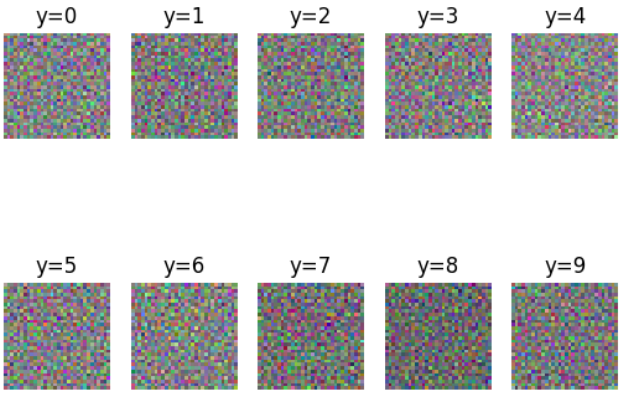


(d) $\lambda = 1,\ \eta = 0.001$

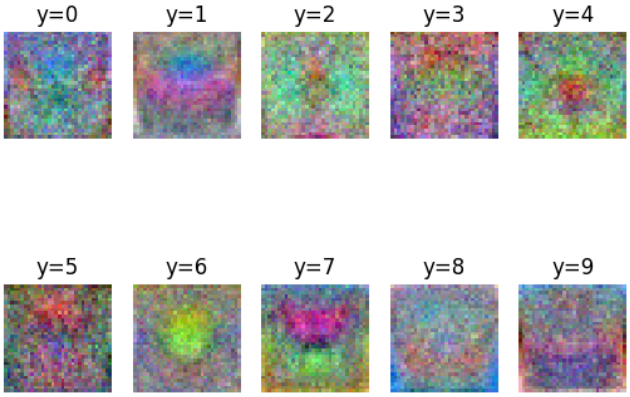
Figure 1: Loss, cost and accuracy of the model during training.



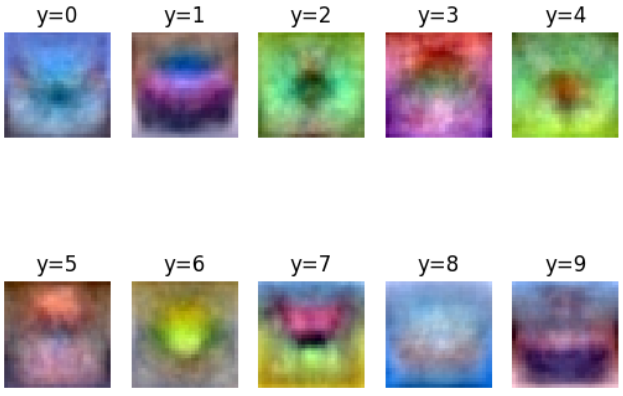
(a) $\lambda = 0,\ \eta = 0.1$



(b) $\lambda = 0,\ \eta = 0.001$



(c) $\lambda = 0.1,\ \eta = 0.001$



(d) $\lambda = 1,\ \eta = 0.001$

Figure 2: Visualization of W after training.