Assignment 2

Léo Noharet - lnoharet@kth.se

April 18, 2022

1 Analytic gradient computation and correctness.

I managed to write the functions to correctly compute the gradient analytically by following the lab assignment instructions and slides from lecture 4. To test that the functions were correctly implemented, I computed the relative error as described in the assignment 1 description, with eps set to 10^{-6}. Then, I also compared the gradients of $\mathbf{W}$ and $\mathbf{b}$ computed numerically and analytically with the help of the assert_almost_equal(actual, desired, decimal) function from the numpy.testing python library. This function verifies that the elements of the matrices *actual* and *desired* satisfy the following: $abs(desired - actual) < 1.5 * 10^{-decimal}$. [Dev] I used this with the parameter decimal set to 8, which proves that the difference is small between the two matrices.
Then, I verified that my mini-batch gradient descent algorithm and gradient computations were correct throught the "sanity check" described in the assignment 2. The result of this can be seen in figure 1

2 Cyclical learning rates, replication of figures 3 and 4.

The graphs shown in figures 2 - 3 replicate figures 3 and 4 from the assignment. Comparing figure 2 and figure 3, we see that the cost, loss and accuracy vary as the learning rate varies. During one cycle, the loss and cost rapidly decline as the learning rate increases, but then plateaus a bit, to then decline a bit more. This could be caused by overfitting to the training data since the regularization term lambda is not optimized. When three cycles are run such as in figure 3, we see that at the end of each cycle when the learning rate is at its maximum, the loss and cost are at their lowest for that cycle, and the accuracy is at its highest for that cycle, illustrating the effect that the learning rate has on the accuracy, cost and loss.
With the parameters used for figure 2, the accuracy on the test data was 45.87% while the parameters used for figure 3 yielded a test data accuracy of 47.31%.

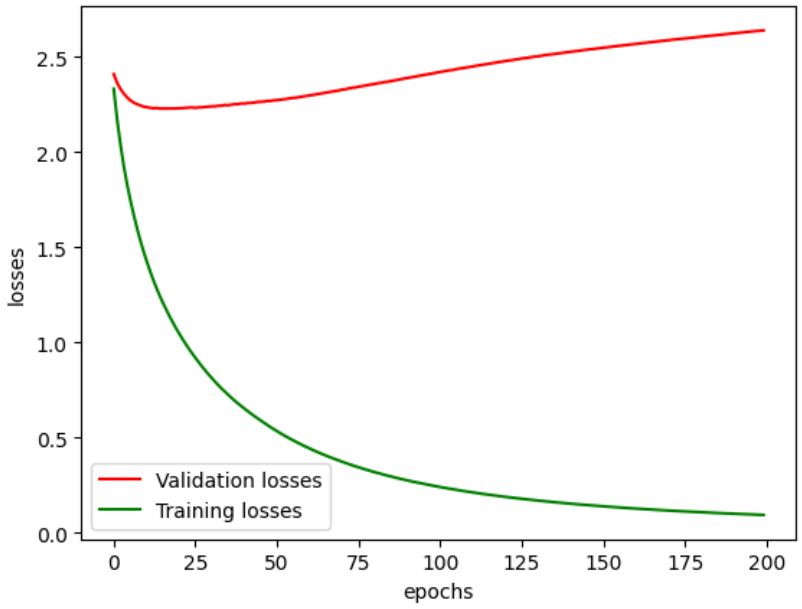

Figure 1: Training and validation loss when overfitting using 100 training examples with lambda=0, n_epochs=200, n_batch=100, eta=.01

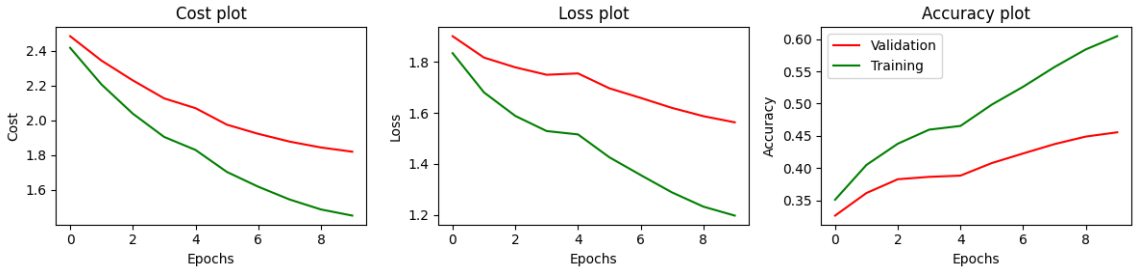

Figure 2: Training and validation cost, loss and accuracy with parameters lambda=0.01, eta min = 1e-5, eta max = 1e-1, and n_s=500, cycles = 1.

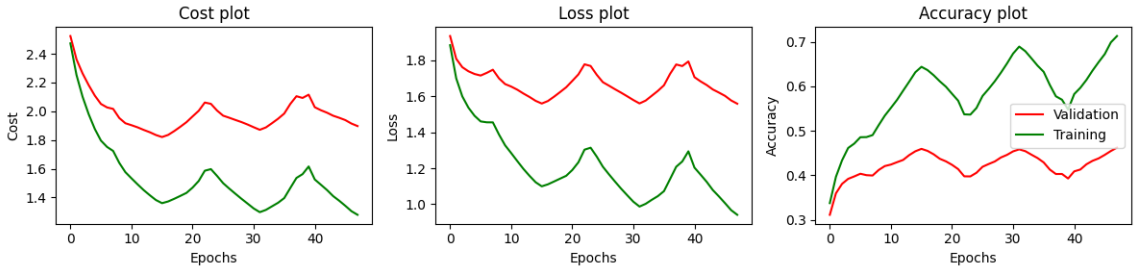

Figure 3: Training and validation cost, loss and accuracy with parameters lambda=0.01, eta min = 1e-5, eta max = 1e-1, and n_s=800, cycles = 3.

3 Coarse search results

For the coarse search, I trained the network with the following lambda values: 10^{-5}, $10^{-4.5}$, 10^{-4}, $10^{-3.5}$, 10^{-3}, $10^{-2.5}$, 10^{-2}, $10^{-1.5}$. For all of the training done during the coarse search, the following parameters were used: $cycles = 3$, $n\_batch = 100$, $n\_s = 2 * floor(n/n\_batch) = 900$. The accuracies of the model on the validation set for each of these lambda values are depicted in figure 4.
The three lambda values that yielded the highest accuracy on the validation set were: 0.00031622776601683794, 0.001 and 0.00031622776601683794 which yielded accuracies on the validation set of 50.8%, 50.2% and 50.6% respectively.

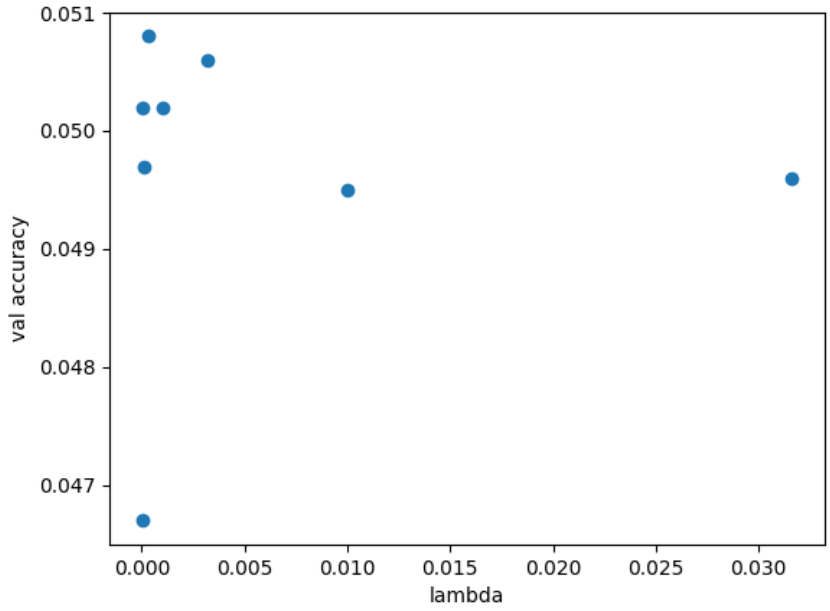

Figure 4: Accuracy of the validation set for lambda values of coarse search

4 Fine search results

For the fine search, I trained the network with the following parameters: $cycles = 3$, $n\_batch = 100$, $n\_s = 2 * floor(n/n\_batch) = 900$. I trained the network 20 times, with 20 different lambda values selected randomly between the range l_min and l_max where l_min is the lambda value that yielded the third highest accuracy on the validation set in the coarse search, i.e. $l\_min = 0.001$ and l_max is the lambda value that yielded the highest accuracy on the validation set in the coarse search, i.e. $l\_max = 0.00031622776601683794$. The 20 lambda values used in the fine search were randomly selected between this range with the use of $lambda = 10^l$ where $l = l_min + (l_max - _min) *$random.uniform(0,1). The accuracy of model on the validation set for each of these lambda values are depicted in figure 5.

The three lambda values that yielded the highest accuracy on the validation set were: 0.0008129568037499997, 0.0017515632216039863 and 0.0013174755122593563 which yielded accuracies on the validation set of 51.4%, 52.4% and 52.4% respectively.

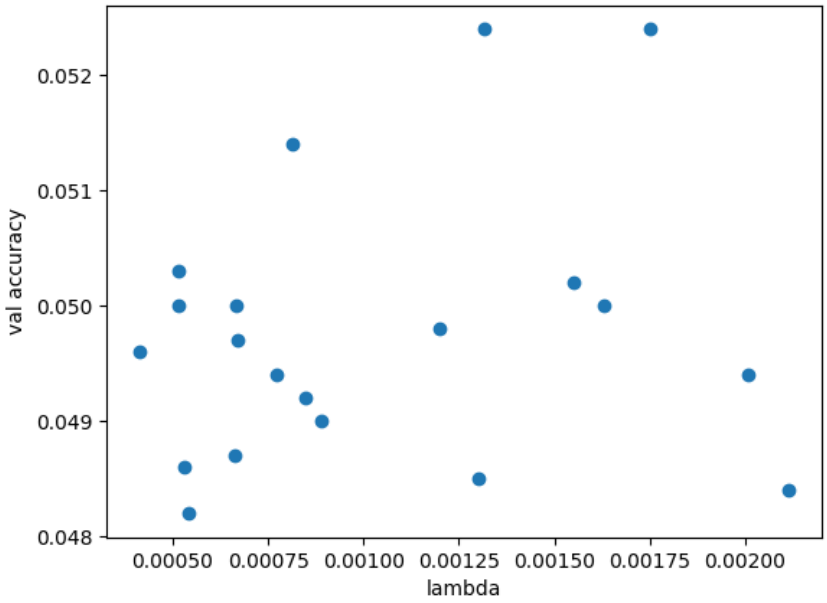

Figure 5: Accuracy of the validation set for lambda values of fine search

5 Best lambda training results

The best found lambda values that the fine search yielded are $lambda = 0.0017515632216039863$ and $lambda = 0.0013174755122593563$ which both yielded an accuracy of 52.4% of the validation data set. When training with all the provided training data but 1000 images, for three cycles, with parameters stepsize = 980 and batch size = 100, an accuracy on the test data of 52.19% was achieved. The loss, cost and accuracy of the training data and validation data with $lambda = 0.0017515632216039863$ is shown in figure 6.
Further, setting $lambda = 0.0013174755122593563$ yielded an accuracy on the test data of 52.21%. As can be seen on figure 6, the difference in loss, cost and accuracy between the training and validation data is smaller when using an optimized lambda, which shows that we are not overfitting as much the model to the training data.

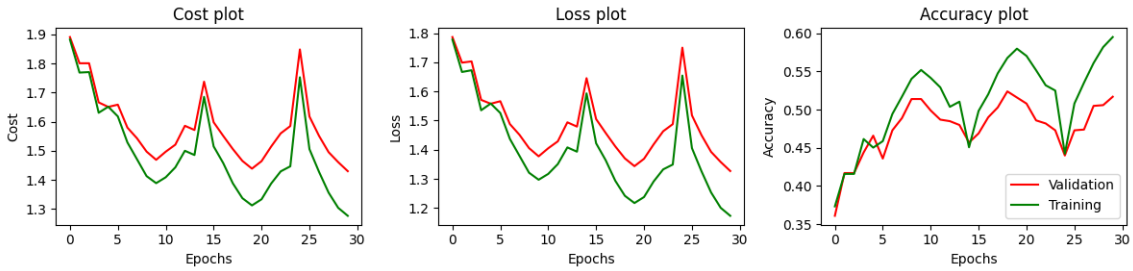

Figure 6: Training and validation cost, loss and accuracy with parameters lambda=0.0017515632216039863, eta min = 1e-5, eta max = 1e-1, and n_s=980, cycles = 3.

References

[Dev] Numpy Developers. numpy.testing.assert_almost_equal. `https://numpy.org/doc/stable/reference/generated/numpy.testing.assert_almost_equal.html`. Accessed: 2022-04-04.