Assignment 1 Report

Adriana Ixba

DD2424 Deep Learning

Josephine Sullivan

Implementation of Gradients

I was able to successfully compute the gradient, using the following equations from Lecture #4 notes:

Forward Pass:

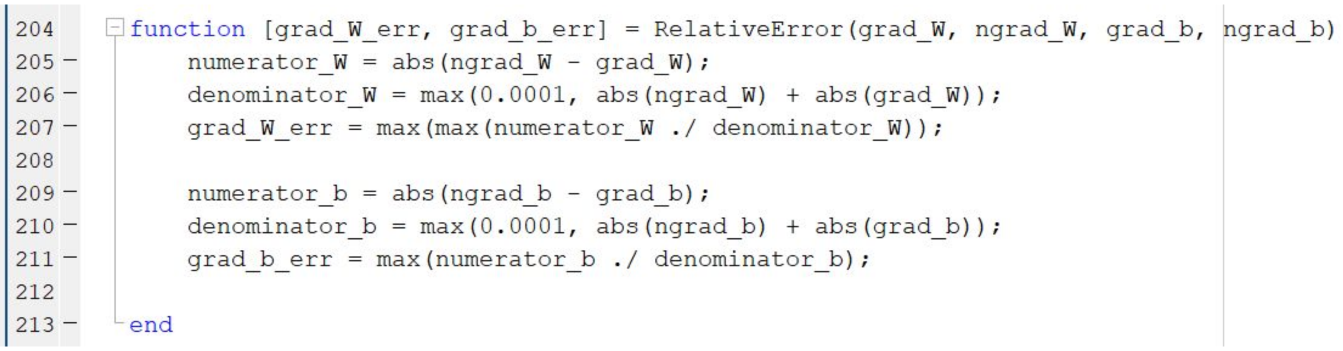
1. $P_{batch} = Softmax(WX_{batch} + b1^T{}_{n_b})$

Backward Pass:

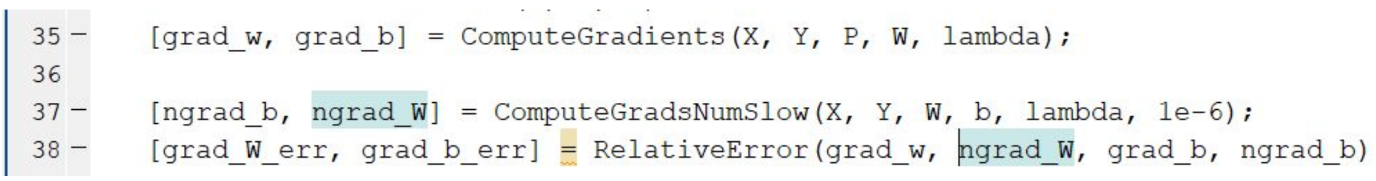
1. $G_{batch} = -(Y_{batch} - P_{batch})$
2. $\partial L/\partial W = 1/n_b G_{batch} X^T{}_{batch}$
3. $\partial L/\partial b = 1/n_b G_{batch} 1_{n_b}$
4. $\partial J/\partial W = \partial L/\partial W + 2\lambda W$
5. $\partial J/\partial b = \partial L/\partial b$

To test my numerically computed gradients, I ran the following equation into code to calculate the Relative Error:

$$|g_a - g_n| \div max(eps, |g_a| + |g_n|), \text{ where the error must be } < 1e-6$$

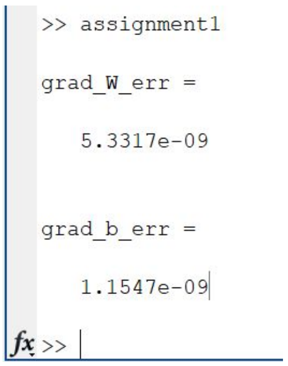


```
function [grad_W_err, grad_b_err] = RelativeError(grad_W, ngrad_W, grad_b, ngrad_b)
    numerator_W = abs(ngrad_W - grad_W);
    denominator_W = max(0.0001, abs(ngrad_W) + abs(grad_W));
    grad_W_err = max(max(numerator_W ./ denominator_W));

    numerator_b = abs(ngrad_b - grad_b);
    denominator_b = max(0.0001, abs(ngrad_b) + abs(grad_b));
    grad_b_err = max(numerator_b ./ denominator_b);

end
```



```
[grad_w, grad_b] = ComputeGradients(X, Y, P, W, lambda);

[ngrad_b, ngrad_W] = ComputeGradsNumSlow(X, Y, W, b, lambda, 1e-6);
[grad_W_err, grad_b_err] = RelativeError(grad_w, ngrad_W, grad_b, ngrad_b)
```

After calculating gradients with my own function as well as with the one provided to test (on the assignment page, ComputeGradsNumSlow, I achieved the following relative errors for the W and b gradients:



```
>> assignment1

grad_W_err =

   5.3317e-09


grad_b_err =

   1.1547e-09

fx >>
```

Results

Total Loss and Costs Results

I show through graphs, the accuracies and costs gathered after each of 40 epochs of the mini-batch gradient descent algorithm. These are the parameters that I used for plotting the figures below.

$$\lambda = 0$$
$$n_{epochs} = 40$$
$$n_{batch} = 100$$
$$eta = 0.01$$

With these parameters, our mini-batch gradient descent algorithm achieved an accuracy of 34.73%. In Figure 1, we can see a smooth exponential-like function. So, we can see that the training itself was somewhat smooth, at least more than that of experiments that will later be described.

Loss Results

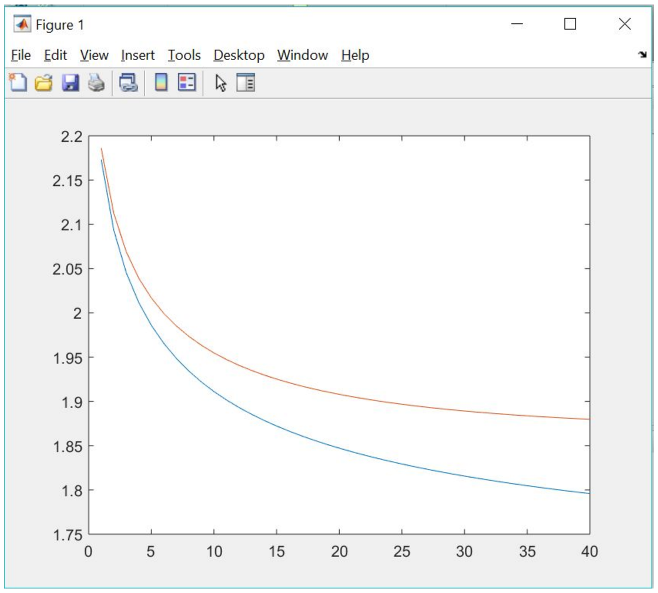


Figure 1. Plot showing the change in Cost the model produces as we iterate through epochs. The red-colored line representing the cost line of the validation set. The blue-colored line represents the cost line of the training set.

Accuracy Results

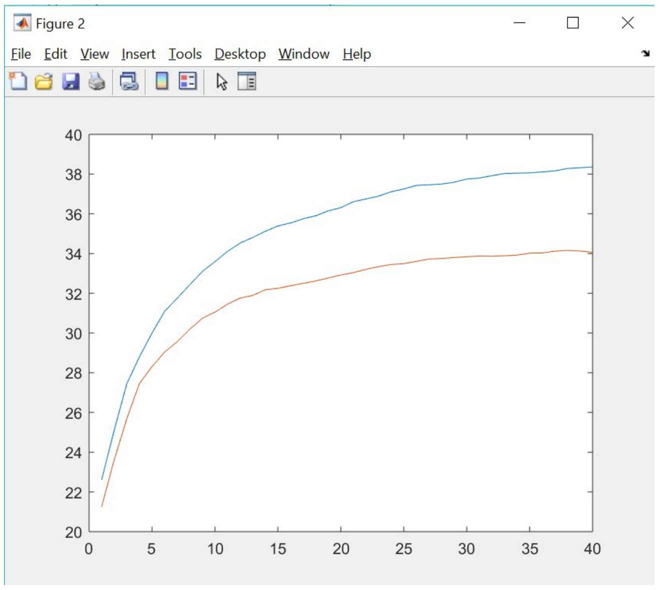


Figure 2. Plot showing the change in Accuracies the model produces as we iterate through epochs. The red-colored line representing the accuracy line of the validation set. The blue-colored line represents the accuracy line of the training set.

The Weight Matrix Results

Next, we analyze a visualization of the weight matrix, $WStar$. We do this using 4 different sets of parameters, each experiment with differing parameters are denoted by Experiment A, Experiment B, Experiment C, and Experiment D.

Experiment A:

$$\lambda = 0$$
$$n_{epochs} = 40$$
$$n_{batch} = 100$$
$$eta = 0.1$$

In this experiment, we achieved an accuracy of **28.33%**. And produced the following visualization for $WStar$.

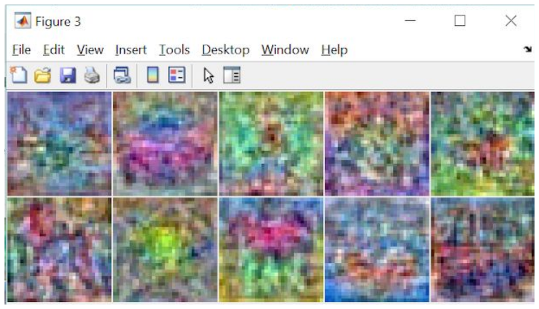


Figure 3. Visualization for Experiment A.

Experiment B:

$$\lambda = 0$$
$$n_{epochs} = 40$$
$$n_{batch} = 100$$
$$eta = 0.01$$

For Experiment B, we achieved a test accuracy of **34.73%**, the highest percentage of the 4 experiments.

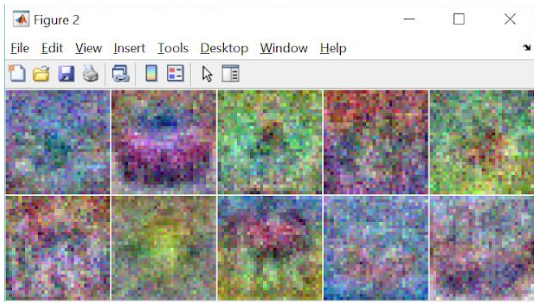


Figure 4. Visualization for Experiment B.

Experiment C:

$$\lambda = 0.1$$

$$n_{epochs} = 40$$

$$n_{batch} = 100$$

$$eta = 0.01$$

For Experiment C, we achieved a test accuracy of **31.88%**.

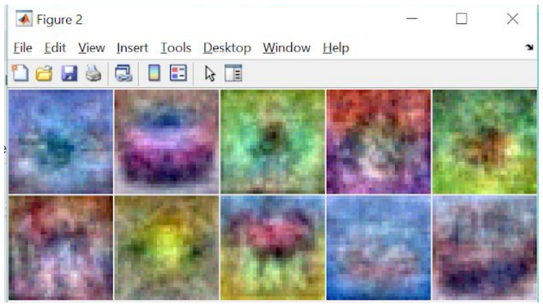


Figure 5. Visualization for Experiment C.

Experiment D:

$$\lambda = 1$$

$$n_{epochs} = 40$$

$$n_{batch} = 100$$

$$eta = 0.01$$

For Experiment D, we achieved a test accuracy of **26.76%**.

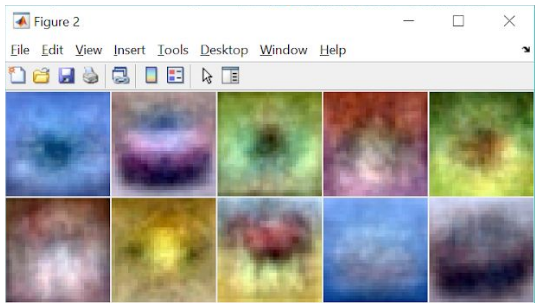


Figure 6. Visualization for Experiment D.

Experiments A and B have a difference of learning rate of a degree of 10. *Experiment A* has an eta of 0.1 (with an accuracy of 28.33%), while *Experiment B* has an eta of 0.01 (with an accuracy of 34.73%). Their other parameters, n_{batch}, λ, and n_{epochs} are exactly the same. They show a difference in accuracies of 6.4%. In these 2 experiments, the learning rate is essential in greatly improving the learning rate of the model. When plotting an accuracy graph for *Experiment A* (Figure 7), we can see that the graph is a lot less uniform than that of *Experiment B* (Figure 2). This is the case, because the learning rate of *Experiment A* is larger, which means that as our model tries to approach a local minima, during the gradient descent algorithm, weight values are changing greatly. This is not allowing for the model to correctly train itself. Thus, producing a poor network performance and test accuracy.

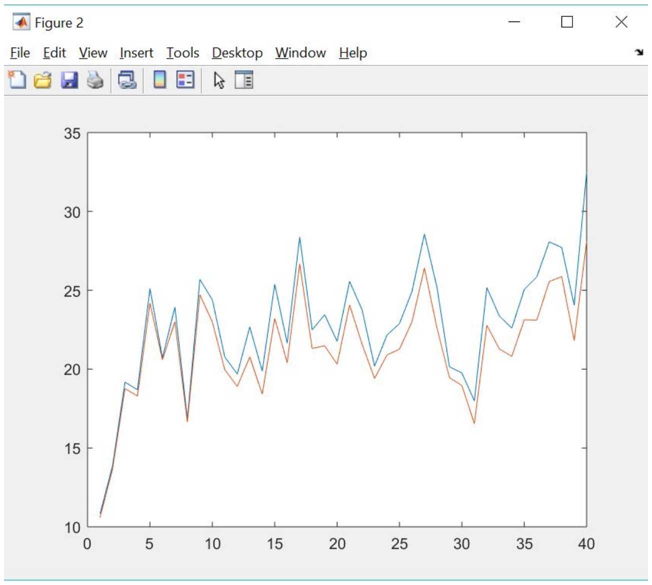


Figure 7. Accuracy plot for Experiment A.

Between Experiments B, C, and D, there are differences in regularization values, λ. Besides the λ value, the rest of parameters are the same: $n_{batch} = 0$, $eta = 0.01$, and $n_{epochs} = 40$. *Experiment B* has a λ value of 0.0, *Experiment C* has a λ value of 0.1, and *Experiment D* has a λ value of 1. Directly looking into the relationship between λ values and accuracy values, we can see that as the λ value increases, the accuracies decrease.

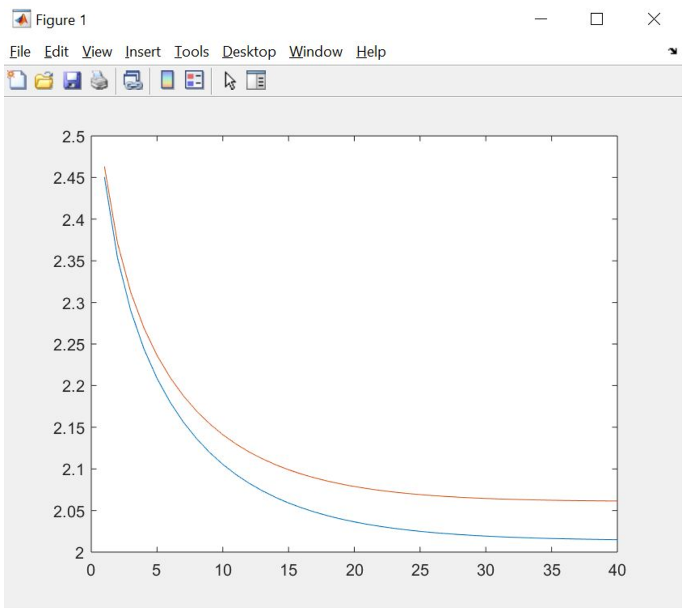


Figure 8. Costs plot of Experiment C.

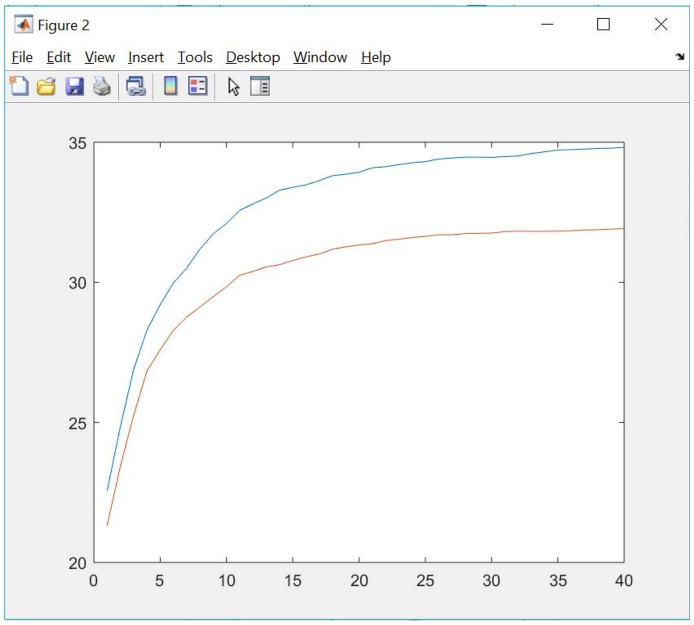


Figure 9. Accuracy plot of Experiment C.

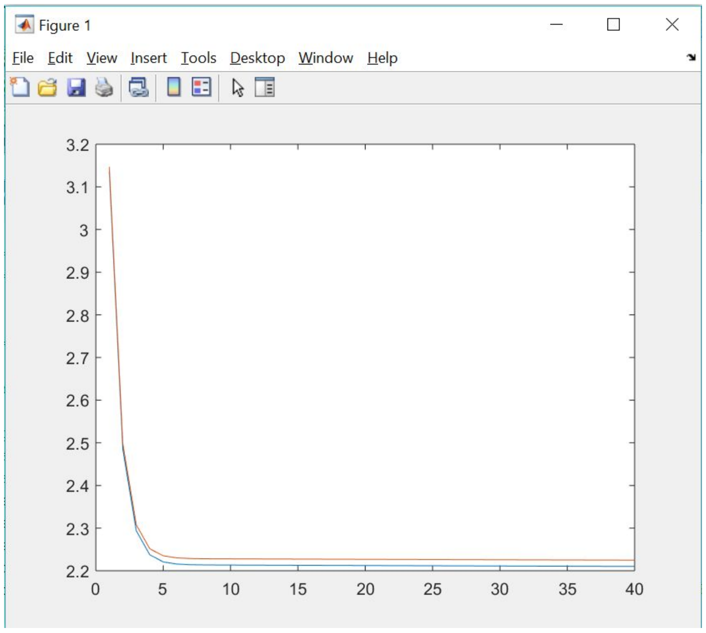


Figure 10. Cost plot for Experiment D.

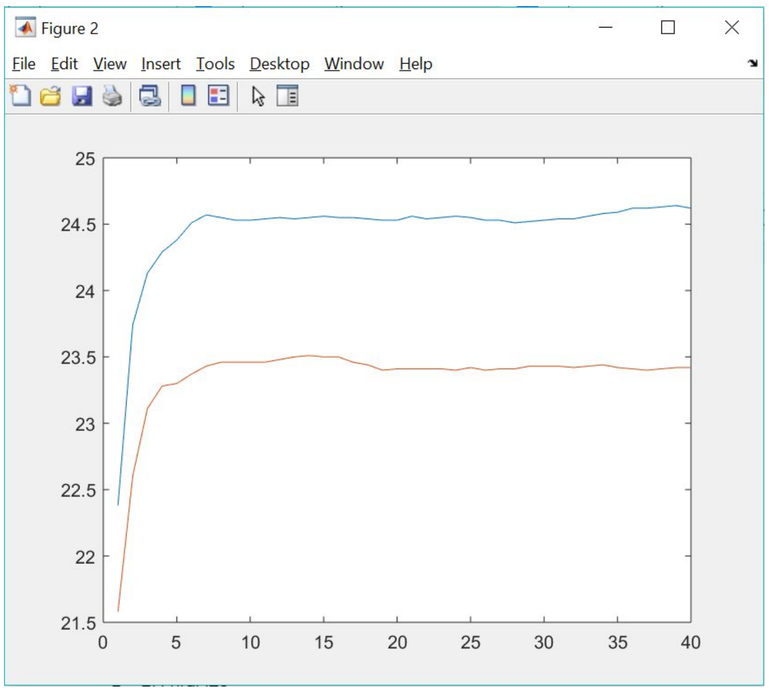


Figure 11. Accuracy plot for Experiment D.

We can see that as regularization increases, the model reaches a peak cost and accuracy much quicker. Figures 10 and 11 show that by epoch 5, the model has reached a somewhat stable accuracy of 24.5%, and a stable cost of about less than 2.1. Compared to *Experiment B*, on the other hand, shows smoother plots (Figure 1 and 2), which achieve better stabled data values, but slower. Because of the quickness in training, the accuracy is a lot lower than models with lower regularization values. And the costs are a lot greater as well. This is because the model is fitting towards the data a lot more and faster. Meaning poor network results. This reflects what was mentioned during lecture that larger regularization promotes fitting towards data, thus heading towards overfitting (Lecture 4).