Assignment 3 Report

Adriana Ixba

DD2424 Deep Learning

Josephine Sullivan

Implementation of Gradients

I was able to successfully compute the gradient. I first constructed a multilayer network. After this, to test the gradients my network produced, I ran the relative error formula. I was able to achieve errors of less than 1e-7.

The following is the Relative Error formula I used to test my computed gradients:

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

The following is the code that I ran for this test:

```
[P, H] = EvaluateClassifier(trainX(1:10, 1:10), W, b);
[dl_dW, dl_db] = ComputeGradients(trainX(1:10, 1:10),trainY(:, 1:10), P, NetParams, lambda, H);

[grad_b, grad_W] = ComputeGradsNumSlow(trainX(1:10, 1:10), trainY(:, 1:10), NetParams, lambda, 1e-5);
[grad_W_err, grad_b_err] = RelativeError(dl_dW, grad_W, dl_db, grad_b)
```

These are the relative error values I achieved compared to GradNumSlow provided in the assignment page.

```
>> assignment3
relative error: 2 layers

grad_W_err =

  1×2 cell array

    {[1.4860e-09]}    {[2.1143e-07]}


grad_b_err =

  1×2 cell array

    {[3.2266e-09]}    {[5.5099e-09]}
```

```
relative error: 3 layers

grad_W_err =

  1×3 cell array

    {[1.1407e-08]}    {[3.3696e-08]}    {[2.8403e-08]}


grad_b_err =

  1×3 cell array

    {[3.7200e-09]}    {[7.3588e-10]}    {[1.5169e-09]}
```

```
relative error: 4 layers

grad_W_err =

  1×4 cell array

    {[4.9613e-08]}    {[1.1596e-08]}    {[1.5091e-08]}    {[1.7269e-07]}


grad_b_err =

  1×4 cell array

    {[4.1826e-09]}    {[1.1904e-09]}    {[1.5964e-08]}    {[1.4091e-09]}
```

Finally, after implementing Batch Normalization, I went on to test my gradients compared to GradNumSlow. This time, my NetParams included the use_bn boolean. The following is the code I ran:

```
[W, b, gammas, betas] = InitializeParametersBN(K, m, d);
NetParams.W = W;
NetParams.b = b;
NetParams.gammas = gammas;
NetParams.betas = betas;
NetParams.use_bn = true;

fprintf("Evaluating Classifier... \n")
meta = EvaluateClassifierBN(trainX(:, 1:5), NetParams);
fprintf("Computing gradients... \n");
[dl_dW, dl_db, dl_dgammas, dl_dbetas] = ComputeGradientsBN(trainX(:, 1:5),trainY(:, 1:5), meta, NetParams, lambda);
fprintf("Computing gradientsNumSlow... \n");
[grad_b, grad_W, grad_gam, grad_beta] = ComputeGradsNumSlow(trainX(:, 1:5), trainY(:, 1:5), NetParams, lambda, 1e-5);
[grad_W_err, grad_b_err] = RelativeError(dl_dW, grad_W, dl_db, grad_b)
[grad_g_err, grad_beta_err] = RelativeError(dl_dgammas, grad_gam, dl_dbetas, grad_beta)
```

When testing my gradients, I knew that they were okay, because I achieved a relative error of $1e^{-6}$ and smaller. So, I continued with my experimentation.

Results

3-Layer Network

I first tested a 3 layer network without Batch Normalization with the following parameters:

$$m = [50, 50]$$
$$cycles = 2$$
$$\lambda = 0.005$$

Training set of 45000, and validation set of 5000.

With these default parameters suggested by the assignment page, I was able to achieve an accuracy of **52.55%**. Figure 1 shows the evolution of the loss of this network. As we can see on our graph, we achieve an appropriate shape for the loss, it slowly decreases to a low loss value.

Loss 3-Layer Network with no BN (52.55% accuracy)

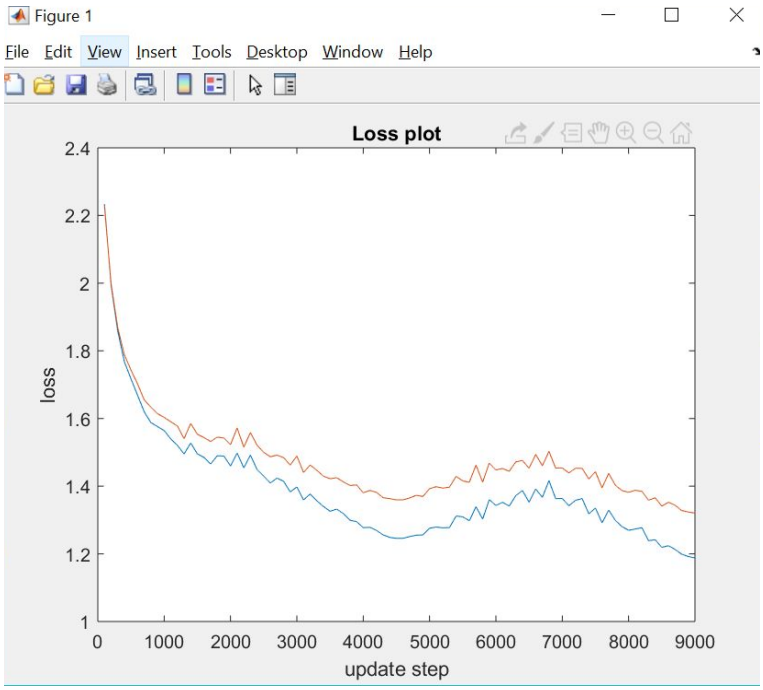

Figure 1. Plot showing what the evolution of the loss a 3-layer model with no batch normalization achieves. The red-colored line representing the cost line of the validation set. The blue-colored line represents the cost line of the training set.

Loss 3-Layer Network with BN (52.8% accuracy)

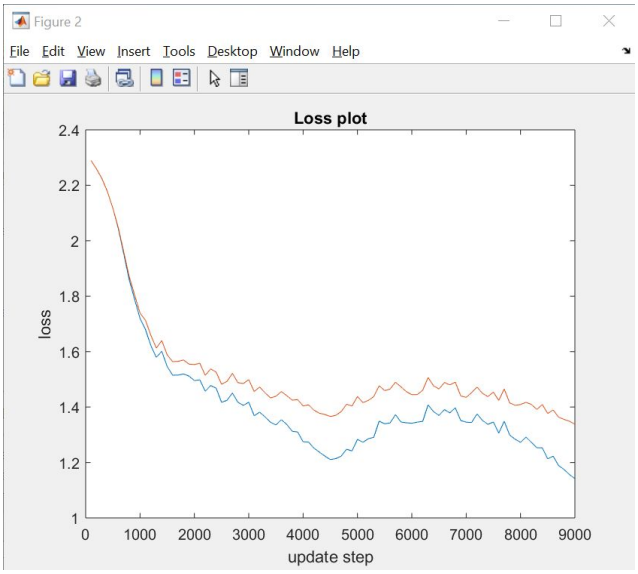

Figure 2. Plot showing what the evolution of the loss a 3-layer model with batch normalization achieves. The red-colored line representing the cost line of the validation set. The blue-colored line represents the cost line of the training set.

Next, I looked at the evolution of the loss function on this same network, except incorporating Batch Normalization. Figure 2 shows this evolution. Interestingly, the functions look somewhat similar, except when the training and validation set are half way through the update-steps. Figure 2 shows a steeper dip for the validation set, compared its training set, when the model reaches around the 4500th update step. But, we do achieve a higher accuracy of **52.8%**.

6-Layer Network

Next, I tested a 6 layer network without Batch Normalization with the same parameters as the ones described for the 3 layer network, with the following exception of the layer initializations:

$$m = [50,\ 30,\ 20,\ 20,\ 10]$$

With these default parameters I achieves an accuracy of **51.07%**. Figure 3 shows the evolution of the loss of this network.

Loss 6-Layer Network without BN (51.07% accuracy)

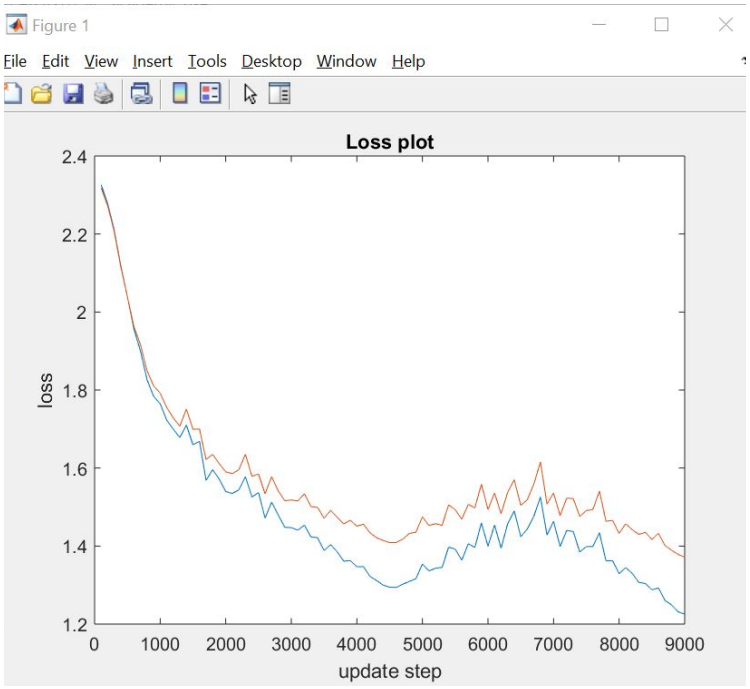

Figure 3. Plot showing what the evolution of the loss a 6-layer model without batch normalization achieves. The red-colored line representing the cost line of the validation set. The blue-colored line represents the cost line of the training set.

Loss 6-Layer Network with BN (51.33% accuracy)

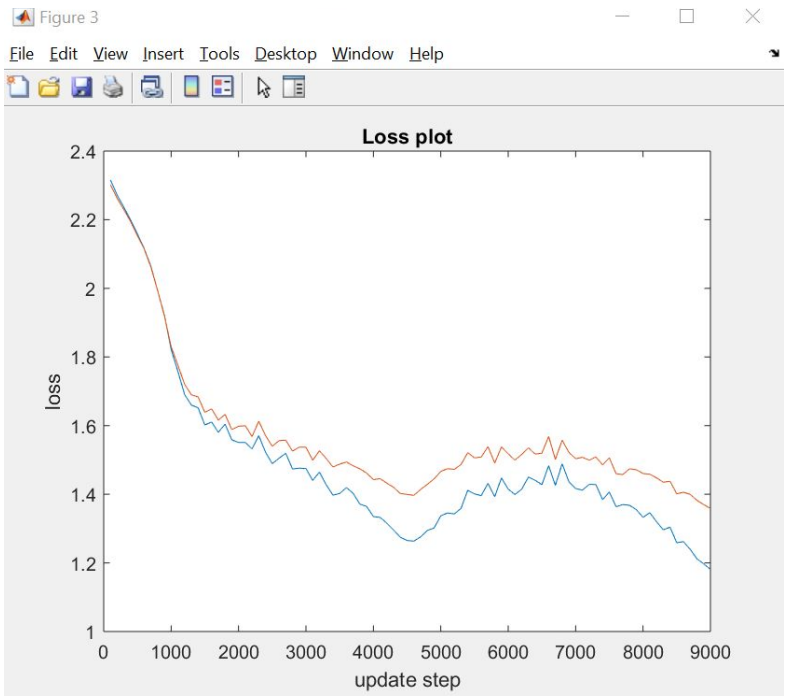

Figure 4. Plot showing what the evolution of the loss a 6-layer model with batch normalization achieves. The red-colored line representing the cost line of the validation set. The blue-colored line represents the cost line of the training set.

We can see that as the number of layers increases, our accuracies decrease, but one thing stays constant: the networks with Batch Normalization have better accuracies.

Coarse Search

I conducted a coarse search and a finer coarse search. First, I looked for lambdas between $1e^{-5}$ and $1e^{-1}$. The best accuracy I achieved with this was **51.07%**. And the lambda that corresponds was **0.09134**.

Coarse-Search ($1e^{-5}$ - $1e^{-1}$)

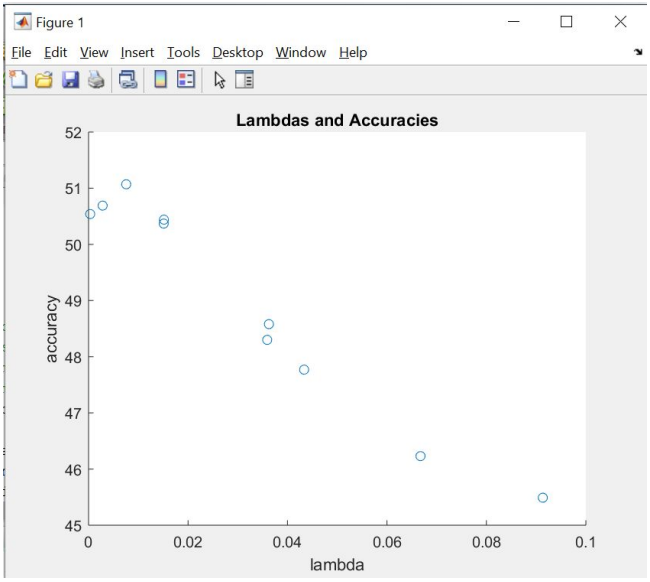

Figure 5. Scatter plot showing the accuracies found for the lambdas between $1e^{-1}$ and $1e^{-5}$.

Finer Coarse-Search ($1e^{-5}$ - $1e^{-2}$)

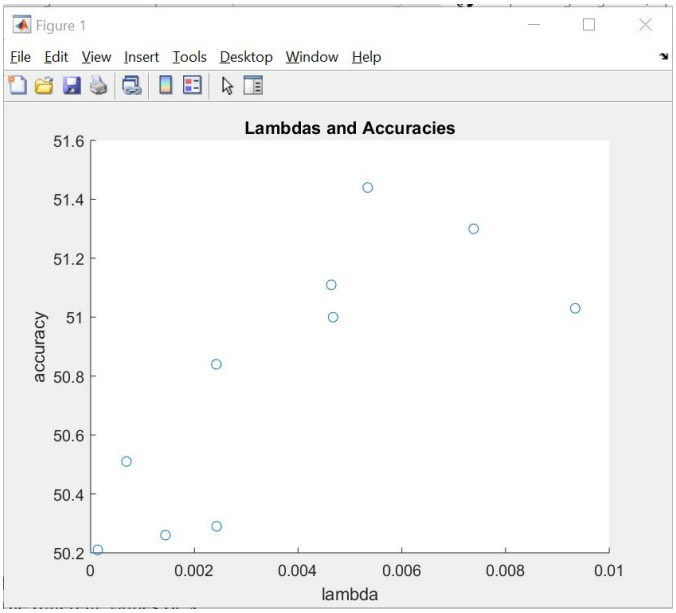

Figure 6. Scatter plot showing the accuracies found for the lambdas between $1e^{-2}$ and $1e^{-5}$.

Next, I conducted a finer coarse-search between $1e^{-5}$ and $1e^{-2}$. This heeded a max accuracy of **51.44%** with a lambda of **0.009343**.

Sensitivity to Initialization

Finally, as suggested by the assignment page, I took a look at experimenting on initialization with and without Batch Normalization, and seeing its effects to the models' loss functions. For all of these experiments, I used a 3 layer network with a lambda of 0.005 (since this produced the best accuracy of 52.8%). I also conducted these experiments with a test set of 45000 and a validation set of 5000.

Sigma of $1e^{-1}$

First, I started with $1e^{-1}$ with no Batch Normalization. This actually produced one of the highest accuracies I had seen throughout experimentation, **52.99%**. Interestingly, when testing using Batch Normalization, the accuracy lowered to **51.19%**.

Sensitivity to Initialization without Batch Normalization ($1e^{-1}$, 52.99%)

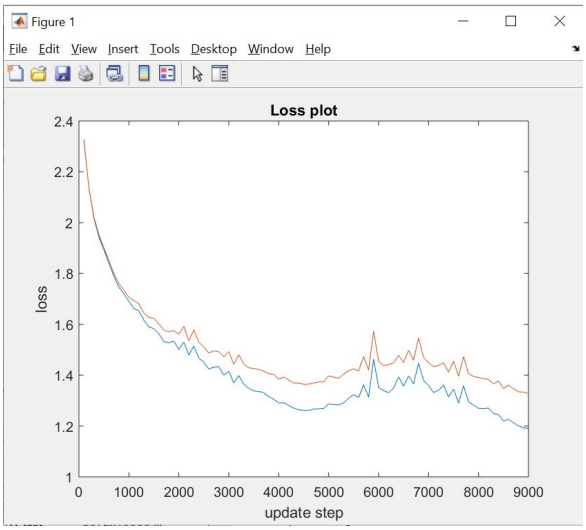

Figure 7. Plot showing the Loss function for experimentation of Sensitivity to Initialization with a sigma of $1e^{-1}$.

Sensitivity to Initialization with Batch Normalization ($1e^{-1}$, 51.19%)

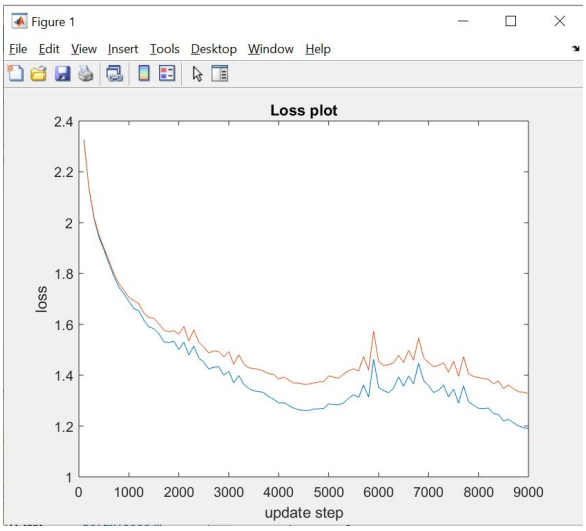

Figure 7. Plot showing the Loss function for experimentation of Sensitivity to Initialization with a sigma of $1e^{-1}$.

Sigma of $1e^{-3}$

For my experiments, initialization was interesting between with and without Batch Normalization.

Sensitivity to Initialization without Batch Normalization ($1e^{-3}$, 48.56%)

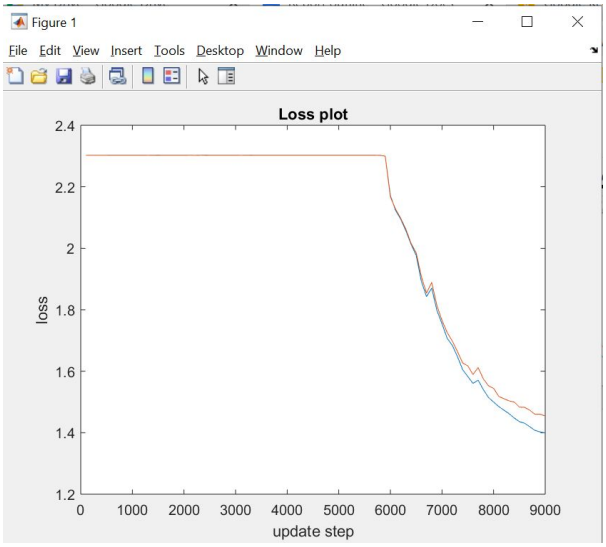

Figure 7. Plot showing the Loss function for experimentation of Sensitivity to Initialization with a sigma of $1e^{-3}$.

Sensitivity to Initialization with Batch Normalization ($1e^{-3}$, 50.69%)

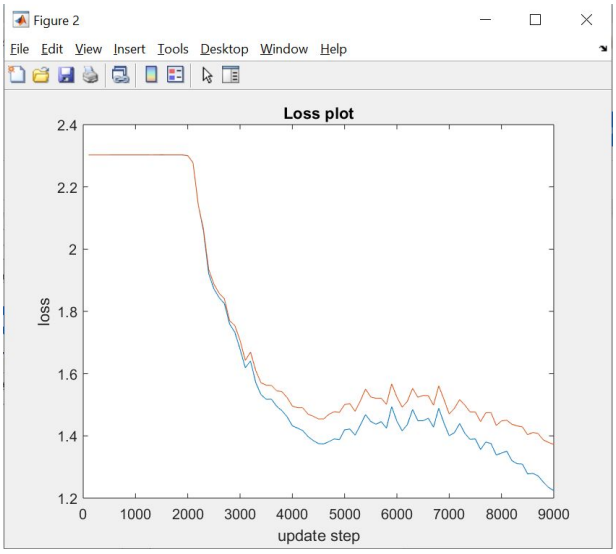

Figure 7. Plot showing the Loss function for experimentation of Sensitivity to Initialization with a sigma of $1e^{-3}$.

Sigma of $1e^{-4}$

Sensitivity to Initialization without Batch Normalization ($1e^{-1}$, 10%)

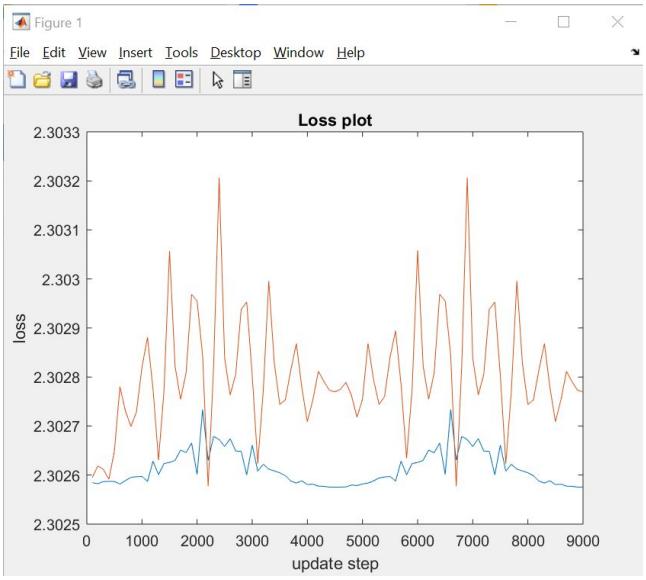

Figure 7. Plot showing the Loss function for experimentation of Sensitivity to Initialization with a sigma of $1e^{-1}$.

Sensitivity to Initialization with Batch Normalization ($1e^{-4}$, 10%)

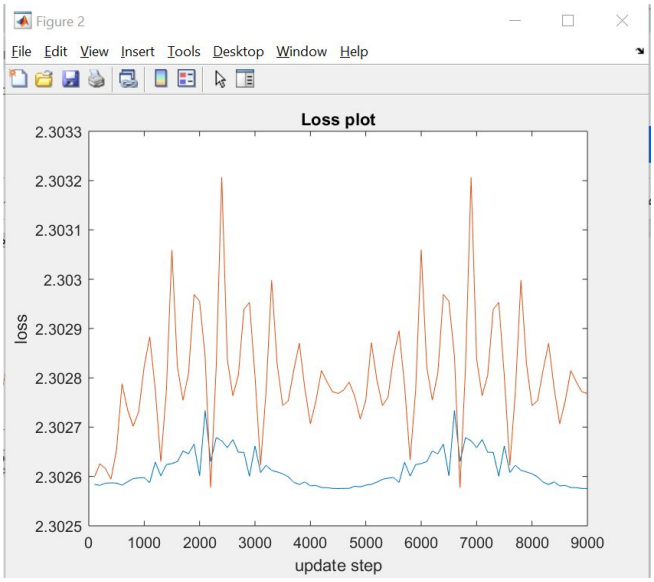

Figure 7. Plot showing the Loss function for experimentation of Sensitivity to Initialization with a sigma of $1e^{-4}$.