**Deep Learning**

**Assignment 1**

**Maxim Katz, 322406604**

**Yuval Levi, 325120384**

**Auxiliary functions:**

We implemented a preprocess function to match the Imported data with format the input of “L\_layer\_model”

def preprocess\_data(train\_x, test\_x, train\_y, test\_y):

The input at each iteration needs to be “flattened” to a matrix of [m,784], where m is the number of samples and we one hot encode the Y data.

def Neural\_Network(train\_X, test\_X, train\_y, test\_y, use\_batchnorm=False, seed=None,l=0):  
Wrapper function that run fit model on training data and predict on validation/test set.

def batch\_size(train\_X, test\_X, train\_y, test\_y, use\_batchnorm=False, seed=None):  
Comparing different batch sizes for finding the optimal batch size.

Specify the batch size:

We run the network for different batch sizes in order to find the best batch size.  
(using the batch\_size function)

[(32, 0.8024), (64, 0.8438), (128, 0.1651), (256, 0.5633), (512, 0.2286), (1024, 0.1742)]

As we can see we got 64 as the best batch size.

**Without BatchNorm:**

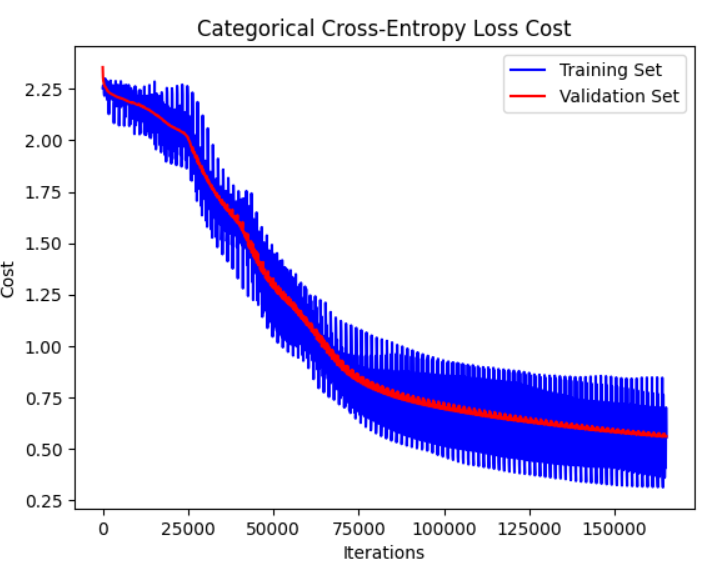
**The final accuracy values for the train, validation and test sets:**

Iterations: 165300; Epochs: 221; Time: 322.44554901123047 seconds

Train accuracy: 0.8424375; Validation accuracy: 0.8375833333333333;

Test accuracy: 0.8438

**Validation and training costs improvement:**

****

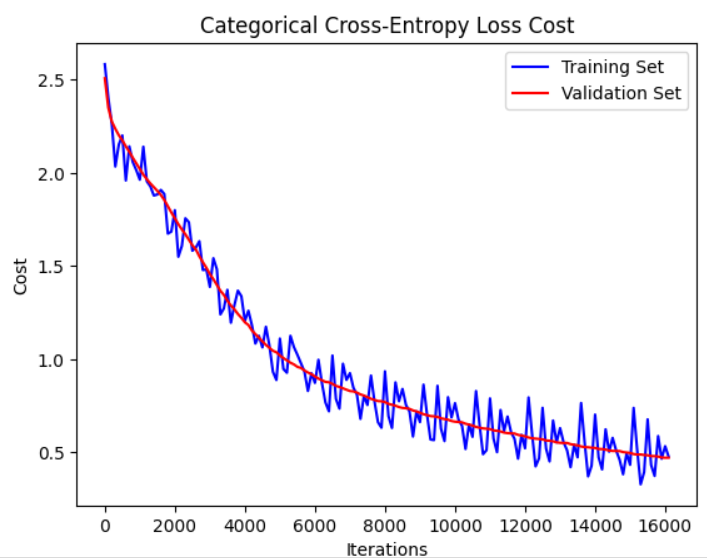
**With BatchNorm:**

**The final accuracy values for the train, validation and test sets:**

Iterations: 16200; Epochs: 22; Time: 31.438477277755737 seconds

Train accuracy: 0.88089583333; Validation accuracy: 0.8824166666; Test accuracy: 0.8834

**Validation and training costs improvement:**

****

**Difference between with and without batch-norm:**

The algorithm run for much less epochs and converged much faster with batch norm, at 22 epochs instead of 221 epochs and 31.4384 seconds instead of 322.4455 seconds.

Also, the accuracy of training, test and validation improved by 4%-5% with much less number of iterations 16200 instead of 165300.

**Adding L2 norm functionality:**

**Changes in the code:**

The main changes, we implemented are in the compute\_cost and linear\_backward functions.

**In compute\_cost:** we changed the signature to (AL, Y, params, r=0) the new cost become the old cost + L2 penalty cost (the squared sum of all the weights in all the layers) as we can see:

Here: m is the number of training examples, L is the total number of layers in your neural network, W[i] is the weight matrix for the i-th layer.

**In Linear\_backward:** we changed the s signature to (dZ, cache, r=0) and the gradient of the weights by adding to change the sensitivity of the weights in gradient descent.

In addition, all the wrapper we added r=0 (for lambda) in the signature of the following functions: **linear\_activation\_backward**, **L\_model\_backward**, **L\_layer\_model**.

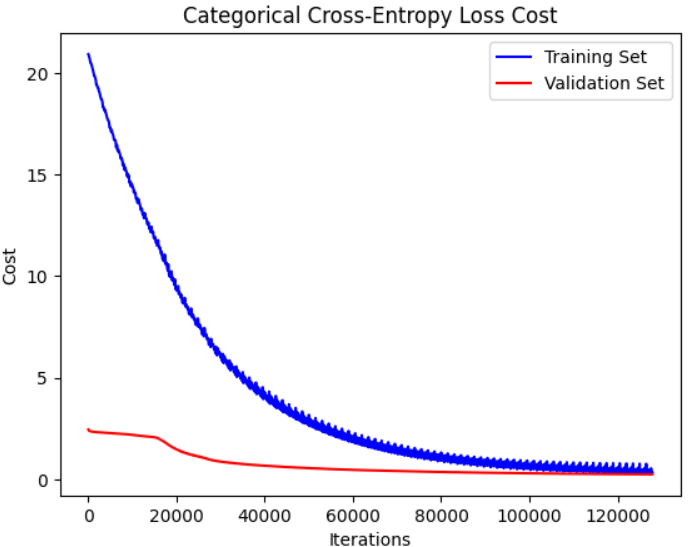
**Without BatchNorm with L2 regularization (r=0.15):**

**The final accuracy values for the train, validation and test sets:**

Iterations: 128000; Epochs: 171; Time: 232.2473976612091 seconds

Train accuracy: 0.945375; Validation accuracy: 0.93133333333; Test accuracy: 0.9363

**Validation and training costs improvement:**

****

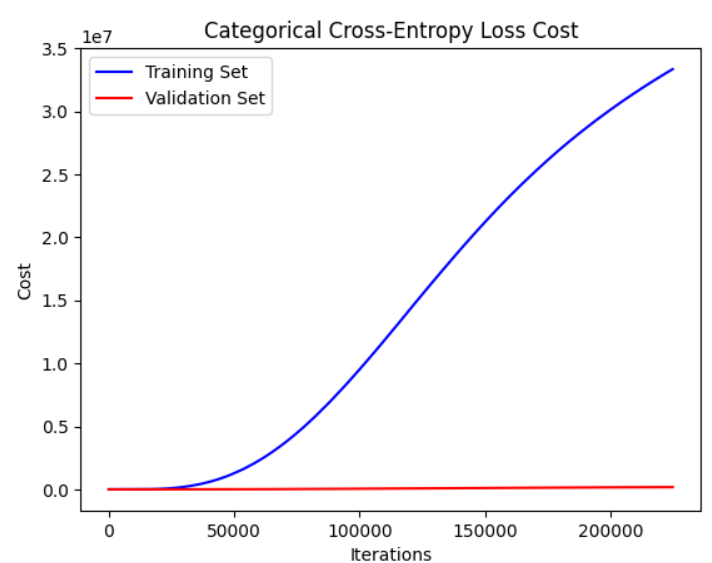
**With BatchNorm with L2 regularization (r=0.15):**

**The final accuracy values for the train, validation and test sets:**

Iterations: 225000; Epochs: 300; Time: 464.0103425979614 seconds

Train accuracy: 0.9254583333; Validation accuracy: 0.92366666666; Test accuracy: 0.9199

**Validation and training costs improvement:**



We discovered that batch norm and L2 regularization don’t work together but can still show good results, which explains the extraordinary results we got while using both.

**Comparing results:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Batch norm | L2 | Train acc | Validation acc | Test acc | Time (sec) | epochs | iterations |
| No | 0 | 0.8424 | 0.8375 | 0.8438 | 322.445 | 221 | 165300 |
| Yes | 0 | 0.8808 | 0.8824 | 0.8834 | 31.4384 | 22 | 16200 |
| No | 0.2 | 0.9453 | 0.9313 | 0.9363 | 137.763 | 171 | 128000 |
| Yes | 0.2 | 0.9254 | 0.9236 | 0.9199 | 464.0103 | 300 | 225000 |

Conclusions:

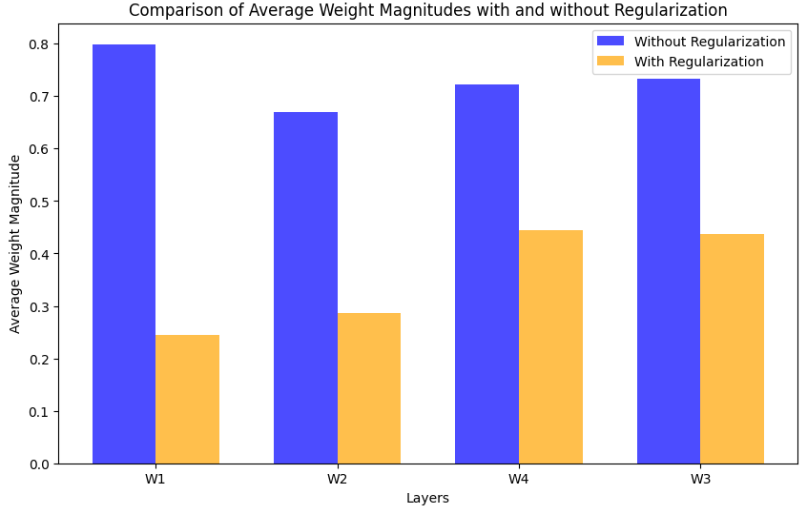
As we can see from the final results using batch normalization produces the best performance in both time and accuracy.

Regarding L2 norm it converges significantly faster than without L2 norm if we don’t use batch normalization.

If we use batch normalization, its final accuracy is slightly lower without L2 (by approx. 9%) but it’s better because it stops earlier.

The green row cannot be compared because batch norm and L2 norm do not work together (it doesn’t regularize the model).

**Comparing weights between with and without L2 norm:**



In the graph here we can see a comparison between the weighted average in each layer of a model.

In the blue bars, it is where we did not use l2 regularization and without batchnorm.

In the orange bars, it is where we used l2 regularization and without batchnorm.

Going through each layer, we checked what was the average of its weights, you can absolutely see that the average of the weights came out clearly higher in the model without l2 (the blue) compared to using it with (the orange). It can be understood that regularization does work and reduces the weights.

In conclusion, we can clearly see that when using l2 we will get smaller weights in a neural network than if we did not use this method.