Binary classification using logistic regression

Practice#3-1

Report by: 완(2019007901)

Changelog for practice#3-1

- Changed the sin function to cos function in both dataset processing for train and test.
- Added ‘sigmoid_dev()’ function and changed the use of ‘sigmoid()’ to ‘sigmoid_dev()’ when calculating dZ1 in back propagation.
- Removed the minus symbol ‘-‘ in ‘cross_entropy_loss()’ function.
- Added matplotlib for plotting cost graph.

Estimated unknown function W & b

For our experiments, we will set the number of samples according to the assignment specifications (M=10000, N = 1000) and the number of iterations K will be set to 5000. After changing the dataset to using cos function, we realized that whether the K is 5000 or 50000, the program keeps yielding accuracy result of ~75%. The reason will be explained in the Discussion section. Therefore, we conclude the best W and b parameters are as below.

```
[w1, b1, w2, b2] = [[[1.31578868]], -117.33321070674467, [[-4.66237314]], 4.002482118914726]
Cost: 0.212847
Cost with n test samples =  0.3107123017625487
Accuracy for 'm' train samples: 74.36%
Accuracy for 'n' test samples: 73.3%

Process finished with exit code 0
```

Empirically determined (best) hyper parameter, α

We will use the same learning rate from the previous assignment, which is 0.01. When we tried values lower that 0.01, we observed that the gradient descent has lower acceleration and if we use value higher than 0.01, the cost keeps oscillation between iterations. Therefore, we conclude that 0.01 is the best learning rate parameter for this program.

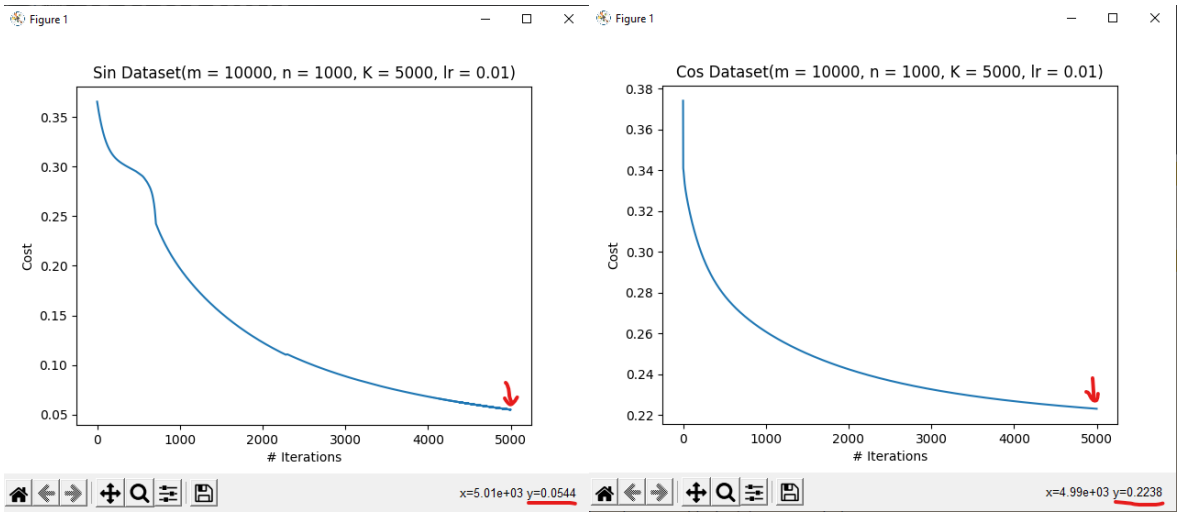
Accuracy

	m=10, n=1000, K=5000	m=100, n=1000, K=5000	m=10000, n=1000, K=5000
Accuracy (‘m’ train samples)	80.0%	78.0%	74.36%
Accuracy (‘n’ test samples)	61.7%	73.5%	73.3%

	m=10000, n=1000, K=10	m=10000, n=1000, K=100	m=1000, n=1000, K=5000
Accuracy (‘m’ train set)	50.17%	50.37%	72.8%
Accuracy (‘n’ test samples)	49.4%	51.0%	73.0%

Discussion

From the accuracy table above, we can see that when we training on cos dataset using the same network, it yields a worse accuracy result than dataset using sin function. After we tried plotting the cost function for both sin and cos dataset, we found something that shows the reason of the low accuracy for cos dataset. The graph is as below.



When training on the sin dataset, we observed that the cost only started to saturated at 0.0544 after 5000 iterations which is pretty low enough to produce a very high accuracy (~99%). Compared to training on cos dataset, somehow the cost started to saturate pretty early at 0.2238 which is still high and therefore causing it to yield lower accuracy (~75%), as proven in the Accuracy section above. Now we have to discuss about the reason behind the early saturation in cos dataset training. If we try to plot the data, the data probably require more complex decision boundary than sin dataset, and perhaps this simple 2-layered caused high bias for our cos dataset. Therefore, to improve the accuracy on cos dataset, we can try to add more layers and units to the network, use different activation function, or try to normalize the dataset first so that the training can be easier and more accurate.

Another problem that we keep encounter since the last assignment is about the immediate cost saturation. Sometimes, if we try to run the program, the cost just died usually at 0.3+ immediately after few interactions. Therefore, we are still trying to find the solution for this issue.