CS4787 Project 1

02/17/2020

Jia Zhang, Shaoqing Jin

Summary:

Based on our plots and tests, we first realized that gradient descent converges really fast, after only around 200 iterations. We also found out that using numpy and matrix operations is much faster than using for loops due to the nice features of modern computers. Furthermore, we could also use a subsample to approximate the real error rate. In conclusion, the techniques implemented and tested in this project can significantly accelerate the process of training and testing.

Part 1.

$$\begin{aligned} TotalLoss &= \sum_{i=1}^{n}[L(h_W(x_i), y_i) + \frac{\gamma}{2}||W||_F^2] \\ &= \sum_{i=1}^{n}[L(h_W(x_i), y_i)] + \frac{n\gamma}{2}||W||_F^2 \end{aligned}$$

The gradient of $L(h_W(x), y)$ is $(softmax(Wx) - y)x^T$. The gradient of the sum of loss function over the whole dataset is equal to the sum of the gradient of loss function over the dataset, which is $\sum_{i=1}^{n}[(softmax(Wx_i) - y_i)x_i^T]$. The gradient of the regularization $\frac{\gamma}{2}||W||_F^2$ is γW . So the gradient of total loss is

$$\sum_{i=1}^{n}[(softmax(Wx_i) - y_i)x_i^T] + n\gamma W$$

A single update step of gradient descent is

$$\begin{aligned} W_{t+1} &= W_t - \frac{\alpha}{n}\sum_{i=1}^{n}\nabla f_i(W) \\ &= w_t - \frac{\alpha}{n}\{\sum_{i=1}^{n}[(softmax(Wx_i) - y_i)x_i^T] + n\gamma W\} \end{aligned}$$

Test Results: LHS=1.1163080401547583

RHS=1.116311487692201

Explanation: After implementing the loss function and gradient of the loss function, we implemented the the test function based on the formula provided in the instruction, with the left hand side replaced by the inner product of V transpose and the gradient, i.e. $< V^T, \nabla f(W) >$. Then we initialized a test example x = (1,2,3), y = (1,0,0) and W = ((1,1,1),(1,1,1),(1,1,1)). Finally, we calculated the both sides with random 3 by 3 matrix V and $\eta = 0.0000000001$. The result shows that the inner product is approximately equal to the derivative.

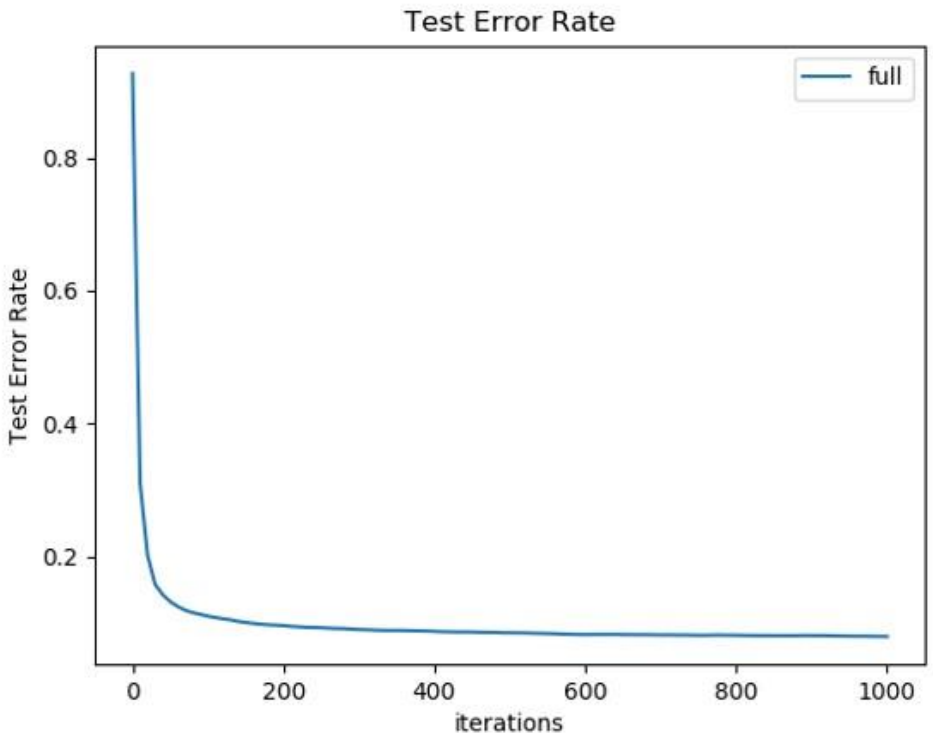
Part 2.

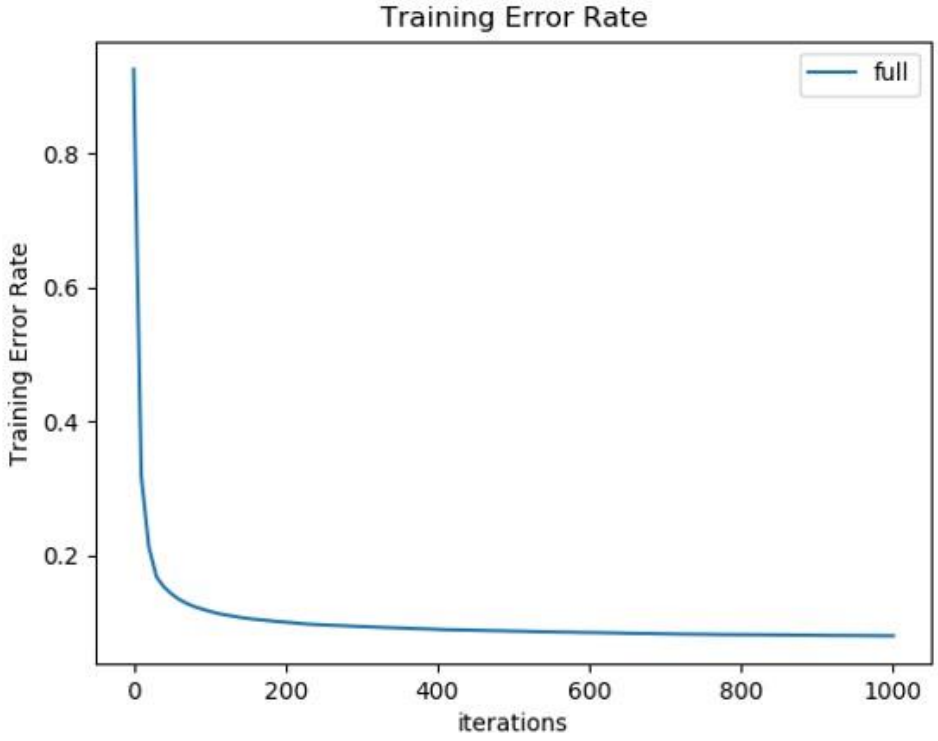
For us, the method using 'for' loops took 51 sec, i.e. 0.19it/sec to run 10 iterations. Therefore, the expected running time for 1000 iterations is approximately 5160 sec, i.e. 1 hour 26 min.

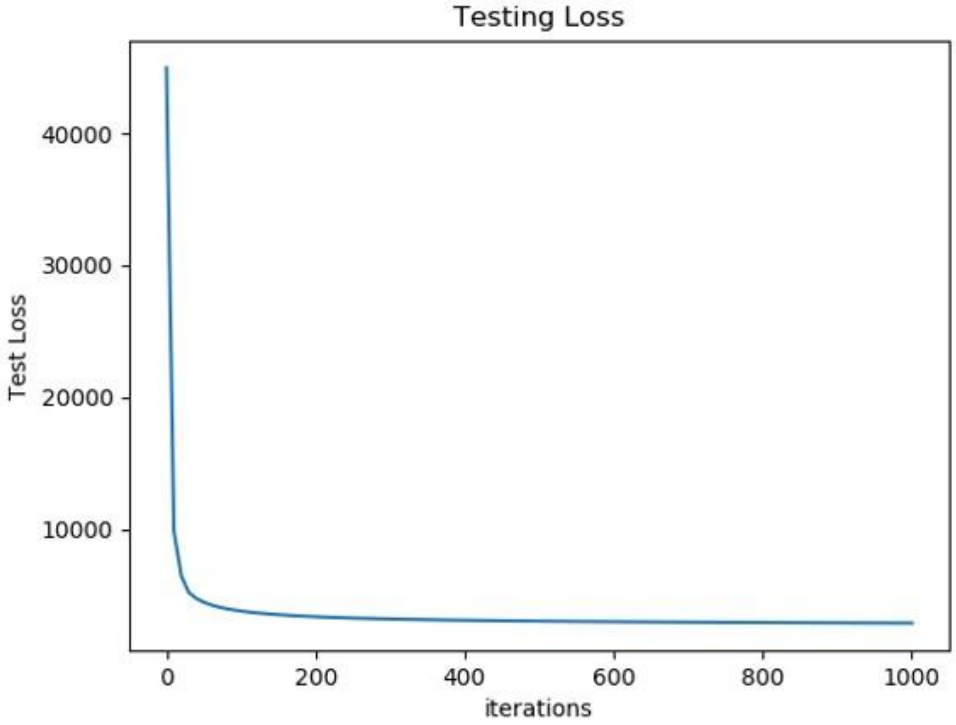
Part 3.

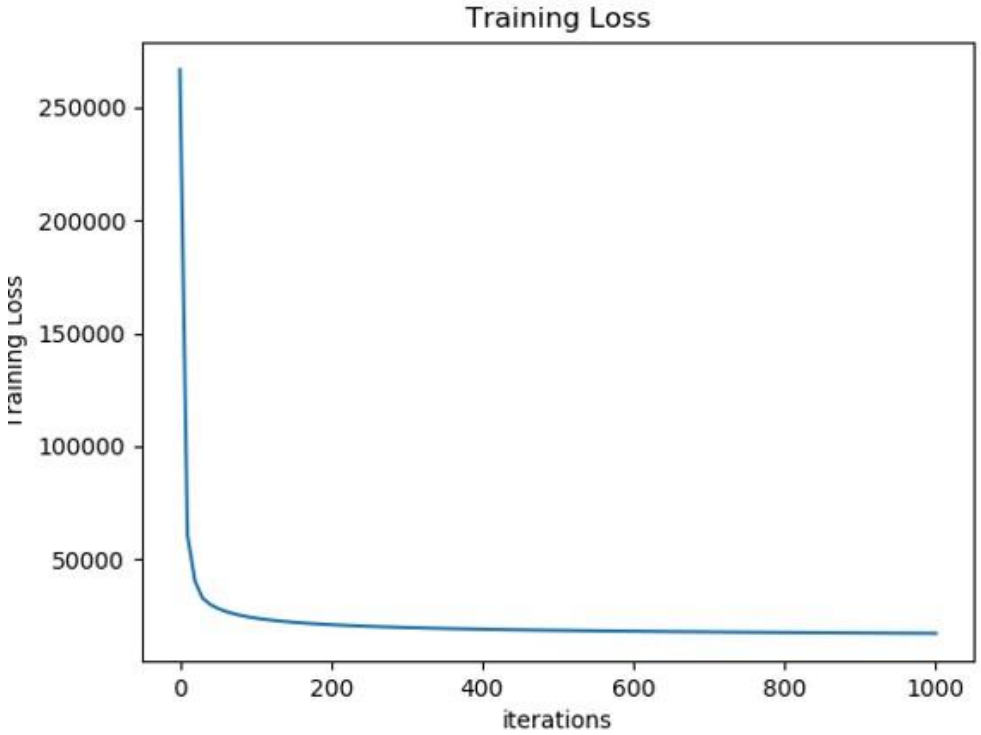
The new method without using for loops took only 2 sec, 4.77it/sec to run 10 iterations, which is about 25 times faster than the method used in Part 2. We think one possible reason is that, as mentioned in the lecture, computers are really good at doing parallel calculations, and numpy matrix operations allow python to utilize this nice feature.

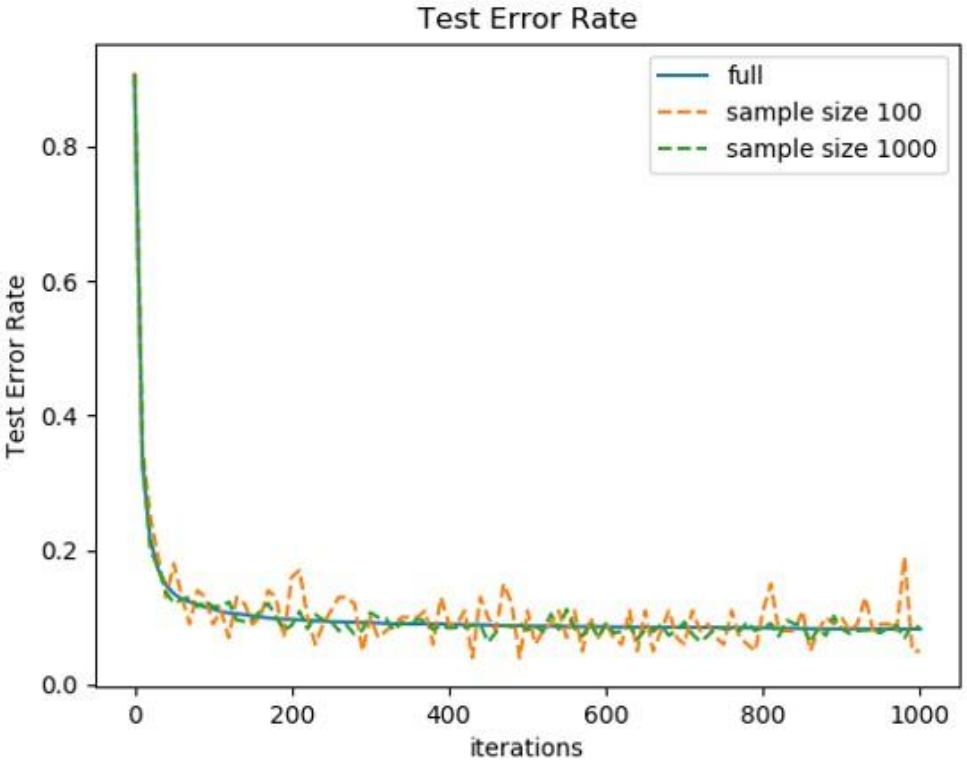
Part 4.

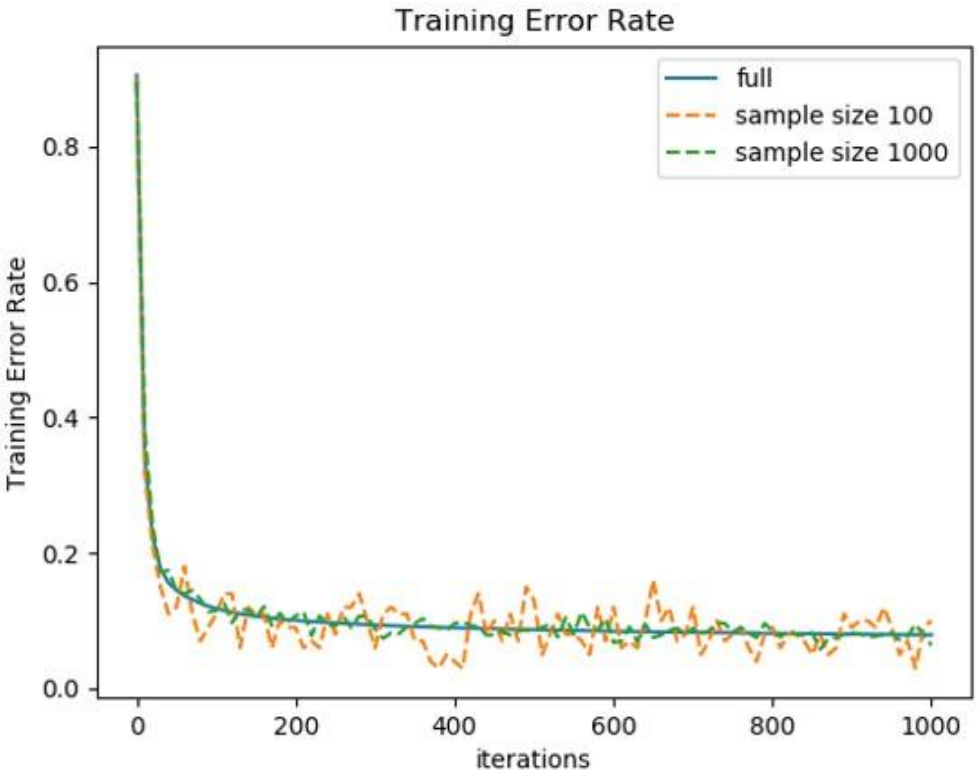

Test Error Rate
full
0.8
0.6
0.4
0.2
Test Error Rate
0
200
400
600
800
1000
iterations


Training Error Rate
full
0.8
0.6
0.4
0.2
Training Error Rate
0
200
400
600
800
1000
iterations


Testing Loss
Test Loss
40000
30000
20000
10000
0
200
400
600
800
1000
iterations


Training Loss
Training Loss
250000
200000
150000
100000
50000
0
200
400
600
800
1000
iterations


Test Error Rate
full
sample size 100
sample size 1000
Test Error Rate
0.0
0.2
0.4
0.6
0.8
0
200
400
600
800
1000
iterations


Training Error Rate
full
sample size 100
sample size 1000
Training Error Rate
0.0
0.2
0.4
0.6
0.8
0
200
400
600
800
1000
iterations

The two plots above show that the test and training error rate calculated from subsamples of size 1000 is sufficient to approximate the real error rate, and it also significantly accelerates the runtime by 25 times for training and 3 times for testing. On the other hand, the subsample of size 100 runs the fastest but its estimation of error is not accurate enough because the estimates fluctuate widely. We know that the sample variance is inversely related to the sample size. Based on the plots above, we believe that subsamples of size 750 are good enough.

Runtime results:

Training:[[00:25<00:00, 3.96it/s]

Test: [00:03<00:00, 26.16it/s]

Training with subsample size 1000: [00:01<00:00, 91.38it/s]

Test with subsample size 1000: [00:01<00:00, 93.00it/s]

Training with subsample size 100 [00:00<00:00, 1423.39it/s]

Test with subsample size 100: [00:00<00:00, 1347.49it/s]

The time complexity of gradient descent is $O(nd)$. So a sample of smaller size n tends to have lower complexity and therefore runs faster.