Tensorflow Report

Group 22

1. Spark and TF comparison:

TensorFlow is specially designed for ML program, and it can work especially well in sparse case. As we tested, in the implementation, before apply the sparse property, we can run one iteration for 1-2 seconds, and after applying the sparse property, we can have 1000 iterations within 6 seconds, which is over 100 times faster. And also TensorFlow supports matrix and vector operation, which is especially useful for ML, and it supports both sync and async mode. Drawbacks: if the feature vector is dense, then TensorFlow may be very slow.

Spark: Supports OLAP queries. But not specified for ML algorithm. It can support more general operations. And it can be fit for iterative algorithm, together with fault tolerance support. Drawbacks: for large data set, with sparse features and models, no specially designed optimization, which may waste a lot of resources. And also it does not support async mode.

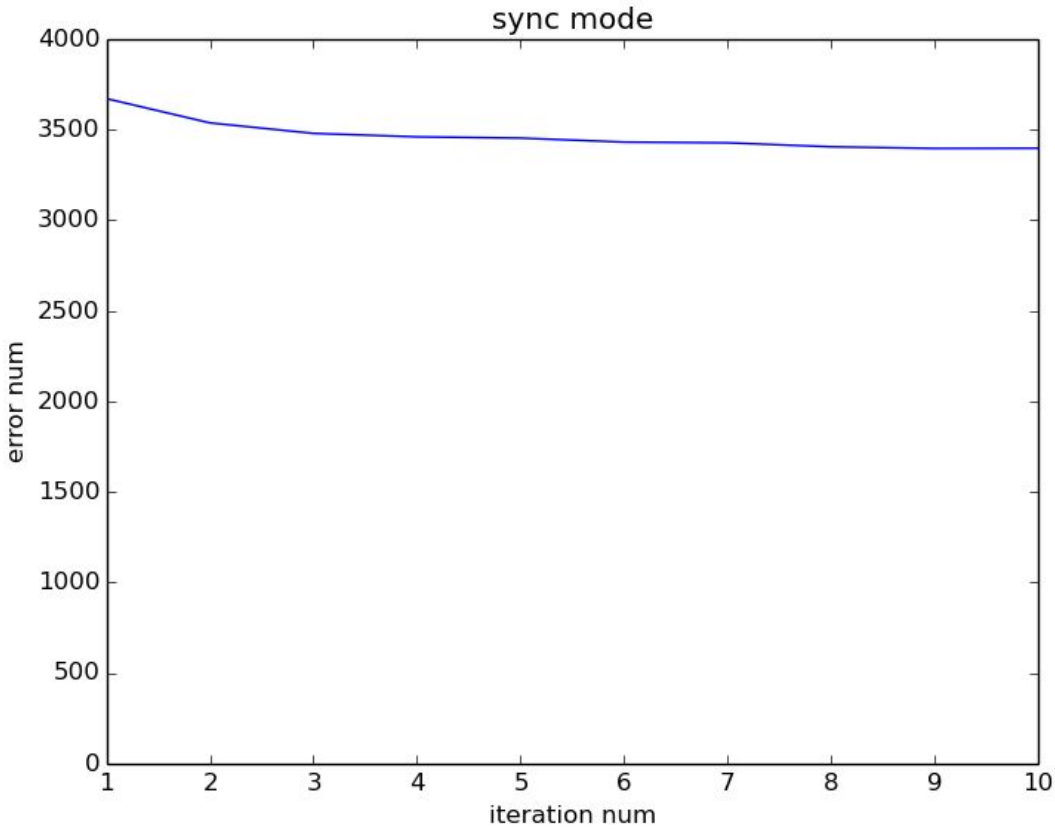
2. Graph:

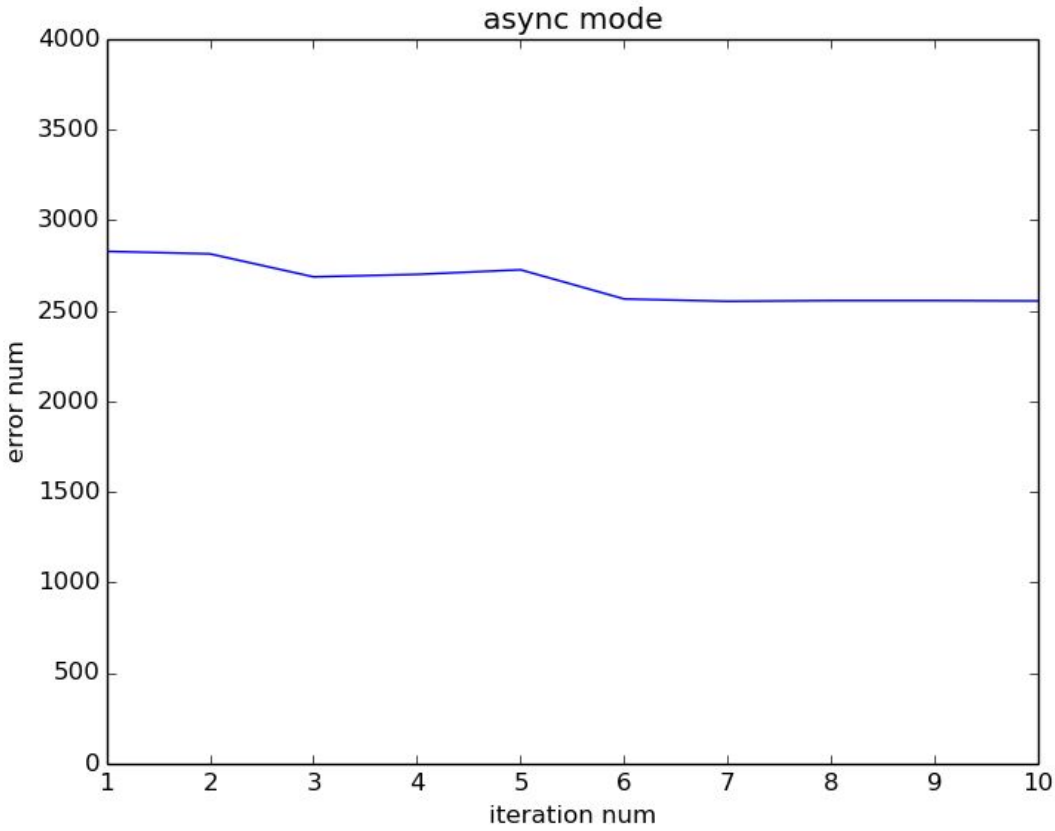
Running settings:

In our implementation, we can run about 1k iterations within 6 seconds for both sync and async, and async can be a little bit faster. The whole iteration requires 2e7, and here we only run 2e4 to save time and it can finish within 15 minutes. And we applied to the test set to get error every 1e3 iterations, and take 1e4 samples as testing set.

The plotting graphs are below. And as we can see, the error is decreasing, and model is converging. And it also verifies that asyn can run faster and converging faster.

	time cost	Final #Error
sync	~ 10 min	3395
async	~ 2 min	2553


sync mode
4000
3500
3000
2500
2000
1500
1000
500
0
error num
1
2
3
4
5
6
7
8
9
10
iteration num


async mode
4000
3500
3000
2500
2000
1500
1000
500
0
error num
1
2
3
4
5
6
7
8
9
10
iteration num

3. Bottleneck:
For bottleneck testing, we simply run sync and async for 5k iterations

	CPU utility	Network Read	Network Write	Disk Read	Disk Write
sync	~ 140%	65 M	65 M	0 M	3 M
async	~ 60%	71 M	71 M	0 M	3 M

According to the above table, we can tell that the bottleneck for sync is CPU, which makes sense because in sync mode, all workers need to wait for the slowest one, and then compute the gradients of data points from five workers all together, requiring higher CPU utility. Async's bottleneck is network, because in async mode, data transmittion through the network is much higher than sync mode. And the disk won't be bottleneck because in our case, we are applying the sparsity, which saves a lot of disk.

Extra Credit:
Implementation: Pretty much the same as before, just need to make some small modifications.
1. Because we need to use sparse representation combined with batch reading, we should use parse_example after batching, instead of using parse_single_exmample as above.
2. In sparse, the indices we want to transform between parameter server and each worker should be all the sparse indices of this batch. We can easily implement this with weight = tf.gather(w, index.values)
3. The batch size is defined by the batch_size variable in code, need to change it manually.

We run a test with 1e4 iterations on sync, and 5e4 iterations on async, and one final test result at last. Apply this on signle data point, 100 as batch size, 1k as batch size, and 10k as batch size respectively. The prediction results are approximately the same, only the time cost varies from each other. And we show it below:
Runtime

iterations per second	single	batch size 100	batch size 1000	batch size 10000
sync	60 sec	46 sec	49 sec	50 sec
async	68 sec	45 sec	44 sec	45 sec

We can tell, from the above table, that when we applied the batch mechanism, the running time gets reduced. But still, even with different batch sizes, time consuming can vary from each other. There should exists a trade-off between batch size and running time. Like in the sync mode, batch size as 100 seems better, while in async mode, 1000 seems preferrable. But also this may have some errors due to small number of iterations, but at least this verifies what we claimed at first.
We didn’t focus on the error rates on the batched size, due to lack of time, and running the whole test set is still time consuming, even though we applied the sparsity optimization. Maybe the batched mechanism can improve the convergence rate after efficient running iterations.