Stochastic Gradient Descent(SGD) Report

----**By QiYao**

Introduction

What's GD and SGD?

As we all know, the gradient descent method is a quite important and often-used method in machine learning field.

For a dataset S={x(i),y(i)}

$$L(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} l\left(\boldsymbol{x}^{(i)}, y^{(i)}, \mathbf{w}\right)$$

where l(x,y,w) is the **loss function** and w is the parameters we want to get,m is the size .

$$l\left(\boldsymbol{x}^{(i)}, y^{(i)}, \mathbf{w}\right) = \frac{1}{2}\left(y^{(i)} - \mathbf{w} \cdot \boldsymbol{x}^{(i)}\right)^2$$

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.

So in algorithm, in each step we will repeat for j=1 to j<n to update all Wj with the whole dataset.

$$w_j = w_j - a\left(\frac{1}{m} \sum_{i=1}^{m}\left(y^{(i)} - \mathbf{w} \cdot \boldsymbol{x}^{(i)}\right)\boldsymbol{x}_j^{(i)} + \lambda \mathbf{w}_j\right)$$

While, **Stochastic Gradient descent (SDG)**, also known as incremental gradient descent, **is a stochastic approximation of the gradient descent** optimization and iterative method for minimizing an objective function that is written as a sum of differentiable functions. In other words, SGD tries to find minima or maxima by iteration. And similar to GD, it also need to update all the W parameters in each step.

$$w_j = w_j - a\left(\left(y^{(i)} - \mathbf{w} \cdot \boldsymbol{x}^{(i)}\right)\boldsymbol{x}_j^{(i)} + \lambda \mathbf{w}_j\right)$$

So in short words, SDG is an improved version of DG. But **the main difference** for these two algorithm is that **during the update process, DG use the whole dataset to update every parameter in each iteration. While SGD only uses a random sample from the dataset each time.**

Why?

Problems exists

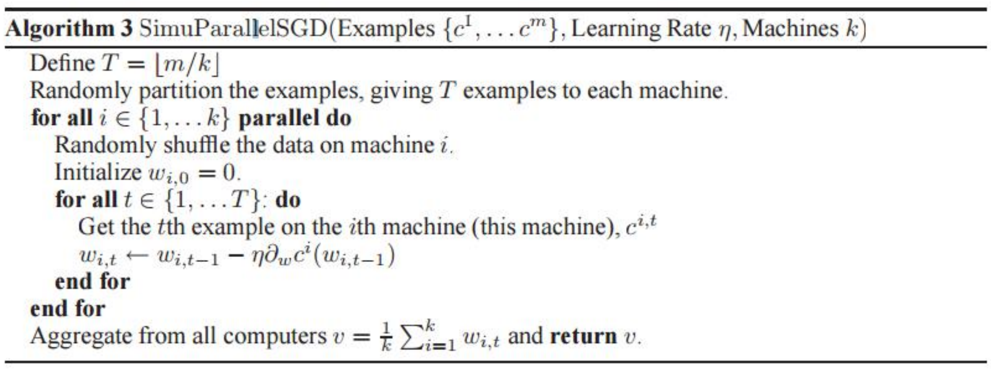
For GD, obviously it's not a suitable method especially when the training dataset is extremely huge. Because in every loop, it will use the whole dataset to calculate the update it will takes a lot of time. So SGD(which only uses only one sample n in an iteration) is currently wide used in ML and becomes much more important.

But what problem does SGD have? First for the algorithm itself, it will need much more iterations than GD does which means its learning rate is really slow. From the other side, we can find that **SGD is a serial and iterative method** which mean every iteration depends on the result of the previous iteration. And therefore, this is not an perfect parallel problem and hard to process it in parallel. And the processing time is quite limited with just one core.

What?

What methods has been proposed?

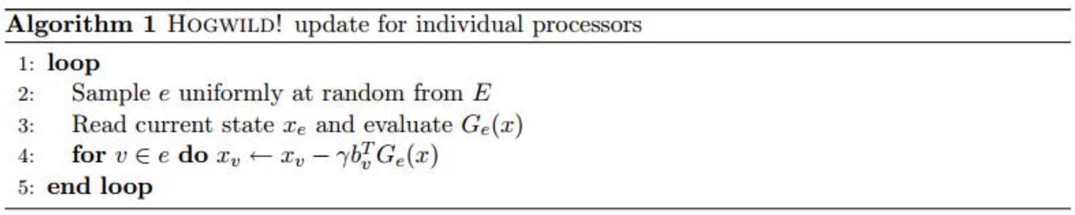
Martin A. Zinkevich,Markus Weimer,Alex Smola and Lihong Li put forward a simple and useful method to accomplish SGD in parallel[1] .And their algorithm shows as below:



```
Algorithm 3 SimuParallelSGD(Examples $\{c^1, \dots c^m\}$, Learning Rate $\eta$, Machines $k$)
  Define $T = \lfloor m/k \rfloor$
  Randomly partition the examples, giving $T$ examples to each machine.
  for all $i \in \{1, \dots k\}$ parallel do
    Randomly shuffle the data on machine $i$.
    Initialize $w_{i,0} = 0$.
    for all $t \in \{1, \dots T\}$: do
      Get the $t$th example on the $i$th machine (this machine), $c^{i,t}$
      $w_{i,t} \leftarrow w_{i,t-1} - \eta \partial_w c^i(w_{i,t-1})$
    end for
  end for
  Aggregate from all computers $v = \frac{1}{k} \sum_{i=1}^{k} w_{i,t}$ and return $v$.
```

What actually they did is, first read the dataset *D* and divide it into *c* different sets. Meanwhile to ensure |D1|=|D2|=|D3|=…=|Dc|=T. And assign *Di* to the *i*th machine. Then for all the partition, parallel do shuffle the data on machine *i.* And give initialize the weight parameters to 0. Then for each machine, do T times iterations to update the weight(All c machines work in parallel). Then aggregate the results from all the computers and get the final results as the average of all results get from all c machines. And it will directly return the average weight vector it got.

Feng Niu, Benjamin Recht, Christopher R´e and Stephen J. Wright[2] put forward a method called Hogwild to make SGD work in parallel. All the CPU can access the data in the memory without adding locks on them. But what it does is assign each CPU with different exclusive parameters w. So you can regard it as every CPU is in charge of different parts. And adding them up is the final result for each iteration.



```
Algorithm 1 HOGWILD! update for individual processors
1: loop
2:   Sample $e$ uniformly at random from $E$
3:   Read current state $x_e$ and evaluate $G_e(x)$
4:   for $v \in e$ do $x_v \leftarrow x_v - \gamma b_v^T G_e(x)$
5: end loop
```

How?

How to do the SGD in parallel with Spark?

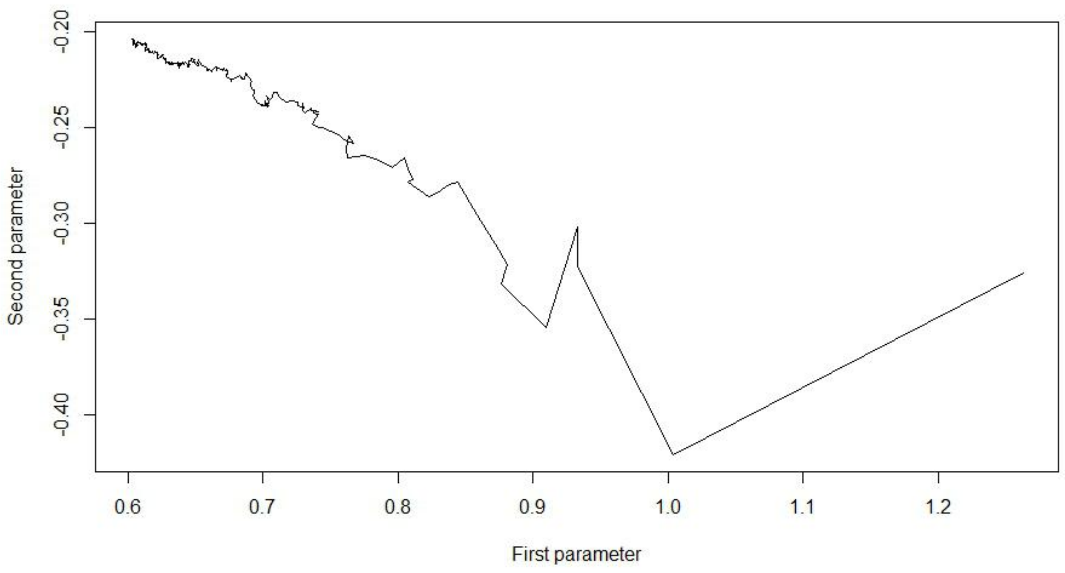
With the work and inspiration of previous work. I think we can achieve the goal by using the first method. Since the communication among between worker and host will take time. So we can actually try to decrease communication frequencies as many as possible.

And try to express in **pseudo code format**:

```
Preprocess the dataset
RDD data=SparkConf sc.textFile(Dataset) //import as RDD
data.repartition();//Repartition the RDD
Shuffle the data
For k=1 to K do(in parallel within K executors)
    Initial $w_{k,0} = 0$
    For all t $\in$ dataset$_k$ do
    $w_{k,t} = w_{k,t-1} - \eta \frac{\partial l(x,y,w)}{\partial w}|_{(x_t, y_t, w_{k,t-1})}$
    End for
End for
Compute the average of models $w = \frac{1}{K}\sum_k w_k$
Return w
```

And in this way, we can have the SGD done in parallel and ensure the communication as less as possible.

How R implements SGD as a benchmark



Where I chose the learning rate as 0.1 and iteration times as 300,and the testing function is y=0.58x-0.21and use rnorm(2) as the initial parameters value. And we can see from this plot graph that the whole parameter generally goes into the same direction.[3]

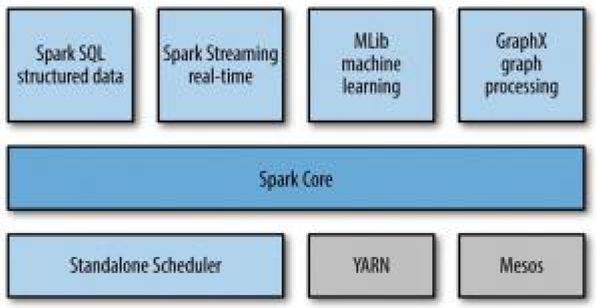
How R implement using RServe on multiple machines?

I think the R implement ideas with RServe&parallel is kinda similar to the principles of spark.

First, we still need to divide the dataset into several parts and host will send these data set partitions to workers. And in each worker it will run the SGD in its own machine with its own assigned data. And the host aggregate the data from the works and get the average w as the final results. And the mainly tool used in R parallel in cluster is SNOW(simple network of workstations) and we can also use “Parallel” packages with parLapply to accomplish parallel computing. While we can also make use of Rserve for communicating R codes among host and workers, this offers the opportunity to have cluster computing in parallel in R .

How is the spark framework implementing the same?

In general,Spark has a various software stack including Spark Streaming, Shark SQL,MLib etc. Spark streaming offers the most basic construction of SparkConf to handle the real-time data streaming. SQL gives spark the ability to operate the SQL while ML makes run ML algorithms. Here we should use MLib and Straming.



And all these ensure a solid environment for various applications.

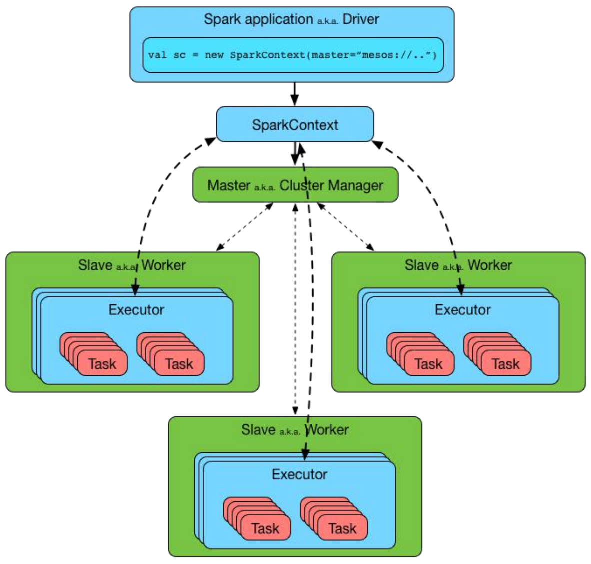
Spark also uses a **host/workers architecture** which means the host has a scheduler to schedule the job and assign all the tasks to the worker node where executors run.And Partition is the minimum operation part in Spark. How many partitions there are, how many tasks there will be.

As for submitting the job, Spark can use local mode, Hadoop YARN mode, Standalone mode as well as Apache Mesos.

For local mode: it just imitate the host-worker with the # of CPU. And if we try to use standalone mode, we can use spark-shell to submit the job to the master node with

```
$./spark-shell --master spark://master:7077--executor-memory XXg
```

And the job driver program is running on the master node and create the SparkContext. And then submit the job to all the worker nodes and get scheduled by the cluster manager process. Below figure shows the architecture in details.[4]



And as for this SGD problem, I think the general architecture can be expressed like: Where we can clarify that when we import data into spark then it becomes a RDD and we **further divided into many partitions** according to the # of the executors and cores in the cluster. Then we **shuffle the data partitions to each executor** and in each executor, we can **run SGD algorithms in all the worker nodes to get the weight results in parallel with its own data partition is got**. **More precisely, we do the whole iteration process to get the W according to the data partition it got in each worker node in parallel**. And finally, the host collect all the results from the workers and get the average of the results it got. Then in this way, we have accomplished SGD in parallel with Spark.

Reference:

[1]Zinkevich M, Weimer M, Li L, et al. Parallelized stochastic gradient descent[C]//Advances in neural information processing systems. 2010: 2595-2603.
[2] Recht B, Re C, Wright S, et al. Hogwild: A lock-free approach to parallelizing stochastic gradient descent[C]//Advances in neural information processing systems. 2011: 693-701.
[3]https://stackoverflow.com/questions/37485138/stochastic-gradient-descent-from-gradient-descent-implementation-in-r
[4]https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-architecture.html