Explore horizontal Fed-Learning for heterogeneous data

Tianning Xu

Abstract

There are many studying on the distributed optimization for horizontal Fed-Learning, where different local machines store different samples. Median-based algorithms are of great interests because of certain robustness against Byzantine workers. However, the convergence of basic median-based algorithm relies on the homogeneous distribution of data samples. In practice, this may not be valid since different machine may store very different samples.

In order to address the convergence problem, Chen et al. [2019] proposes the idea that we add artificial noise to the gradient information of each local machine and then calculate median from this information.

This project wants to understand the non-convergence phenomenon of median-based gradient descent and how the additional noise contributes to the convergence. Besides adding noise, we try another idea of 'quantileSGD' to address the non-convergence issue of median-based GD. Those idea works on the logistic regression and synthetic data.

1 Introduction

1.1 Literature review

Traditional machine learning trains models on the local machines. Recently, many interests develop on the Federal learning problems, because of the increasing requirement for computation resources and the privacy of distributed data. Dean et al. [2012] shows one successful application of distributed learning into neural networks. With the help of abundant distributed resources, the deep network can be trained at the scale of 100 times than previously reported in the literature.

One specific area of interest in distributed learning is sample-partitioned training algorithm, called horizontal federal learning. Both security and robustness become important issues in such problems. Suppose we have M machines and the data-based object is $f_i(x)$ on each machine. The problem can be formed as:

$$\min_{x\in\mathbb{R}^d} f(x) \triangleq \frac{1}{M}\sum_{i=1}^{M} f_i(x) \tag{1}$$

where each local machine (local node) i can only access to its own data. The information from local machines is aggregated at one master machine.

One natural first order optimization algorithm is to average the gradient information from each local machine. However, local machine may not always be stable and reliable since they can be cell phones or personal computers. Consequently, one potential issue is Byzantine failure (Lamport et al. [1982]), where some local machines may send abnormal message to master machine. Several robustness algorithms including SIGNSGD in Bernstein et al. [2018] and MEDIANSGD in Yin et al.

[2018] work on robustness against Byzantine workers. Under some conditions, equivalence of those two methods are proved in Chen et al. [2019].

Algorithm 1: medianSGD

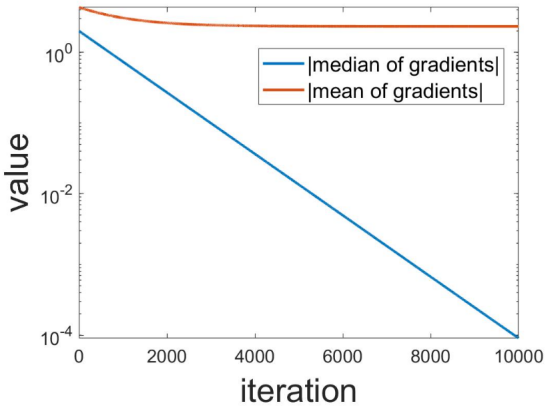
Input: learning rate δ, current point x_t

$g_{t,i} = \nabla f_i(x_t)$ + sampling noise ;

/* The median operation is coordinate-wise median */

$x_{t+1} = x_t - \delta \text{ median}(\{g_{t,i}\}_{i=1}^M)$;

However, another issue is the heterogeneously distributed data. The distribution of data in each local machine may not be identical or similar. In calcification problems, it's possible that different categories of samples store in different local machines. Chen et al. [2019] claims that those robust FL algorithms may fail for heterogeneous data. An one-dimensional toy example is used in Chen et al. [2019] to show this failure: $f(x) = \frac{1}{3}\sum_{i=1}^3 f_i(x) = \sum_{i=1}^3 (x - a_i)^2/2$, with $a_1 = 1, a_2 = 2, a_3 = 10$



(b) Trajectory of MEDIANSGD

Figure 1: failure of medianSGD: toy example in Chen et al. [2019]

From figure 1, even though the median of gradients converges successfully, the mean of gradient does not converge. This is not surprising because median of gradient is used in GD algorithm instead of the mean. If median is not close enough to mean, then non-convergent of mean gradient happens.

To overcome the issue from heterogeneity, Chen et al. [2019] proposes noisy version of medianSGD and signSGD:

Algorithm 2: Noisy medianSGD

Input: learning rate δ, current point x_t, random noise $\epsilon_{t,i}$, noise magnitude b

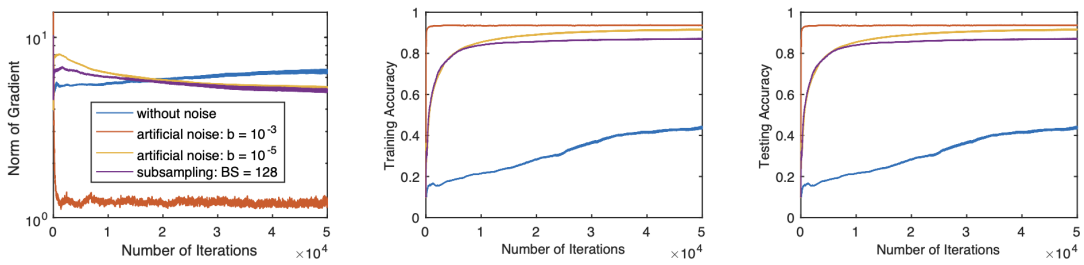
$g_{t,i} = \nabla f_i(x_t)$ + sampling noise + $b\epsilon_{t,i}$;

$x_{t+1} = x_t - \delta \text{ median}(\{g_{t,i}\}_{i=1}^M)$;

where iid Gaussian is one kind of such noise. The idea is that the artificial noise on local gradient information may help reduce the gap between expected median and the mean of the gradient. Adding noise also preserve some nice properties previous methods, such as the low per-iteration communication complexity.

To show the success of noisy medianSGD algorithm, Chen et al. [2019] trains the 2-layer NN on the MNIST data. Each node/machine only contains exclusive data for one or two out of ten

categories, which corresponds to heterogeneity. The convergence from noisy median to mean and the convergence of noisy medianSGD algorithm are proved under some conditions.



(a) Norm of Gradient (b) Training Accuracy (c) Testing Accuracy

Figure 3: MNIST comparison of SIGNSGD with and without noise. We train a two-layer fully connected neural network on MNIST dataset distributed over 10 machines.

Figure 2: The experiment on MNIST in Chen et al. [2019]

1.2 Works in this project

In this project, we focus on the materials in Chen et al. [2019].

1.2.1 Understand the phenomenon

First, we want to have more understanding on how medianSGD fails and how the noisy medianSGD works under heterogeneous data. To name some issues of interest:

- How large is gap between the median of gradient to mean gradient?
- How is the performance of noisy medianSGD distributed training compared SGD in pooled training?
- Besides the testing accuracy, we want to know additional detailed information about the model. For example, for classification problem, would some classes be 'ignored' or be paid less attentions to?

Possibly due to limitation of pages and the black-box property of neural networks, those issues are not very clear in Chen et al. [2019]. We are going to do experiments on synthetic data and one simple model with convex loss function: logistic regression, to get some better understanding.

1.2.2 An alternative idea: 'Quantile-SGD' to overcome heterogeneity issues

To make medianSGD algorithm works on heterogeneous data, adding artificial noise is one solution. It is also mentioned in Chen et al. [2019] that before taking median, sub-sampling the gradient information from each local machine may also help for convergence.

In this porject, we propose and check one heuristic idea: for each iteration, instead of only the coordinate-wise median from local machines, we may use some coordinate-wise quantile gradient and then conduct the gradient descent on master machine. This may reduce the gap between median and median and also preserve some robustness against Byzantine failure. Details will be discussed in section 2.

2 Quantile-SGD algorithm

2.1 Motivations

Chen et al. [2019] claims that 'adding noise' works as perturbing the local gradients. The other mentioned method 'sub-sampling' is also a tool for perturbation. It seems perturbation on local information before aggregating is the key to overcome the non-convergent issue of medianSGD. We may try some other ways that perturb the gradient information before aggregating in the master machine.

In addition, median gradient comes from only one or two machines, which may waste some information compared to mean when data distributed heterogeneously. We want to make use of more gradient information when conducting gradient descent in the master machine.

One natural extension of median is the quantile. The idea is that, besides the median, we can make use of some 'quantile gradient' around the median. The algorithm can be either 1) with certain probability, pick some quantile gradients instead of median or 2) weighted average those quantile gradients.

2.2 Implements: weighted quantile-SGD and random quantile-SGD

Algorithm 3: weighted quantileSGD (one iteration from local machine to master machine)

Input: learning rate δ, current point x_t,
Input: weights $w_1, ..., w_c$, st. $\sum w_j = 1$
$g_{t,i} = \nabla f_i(x_t)$ + sampling noise, i = 1, ..., M ;
from $\{g_{t,i}\}_{i=1}^{M}$, calculate c many candidate coordinate-wise quantiles: $q_1, q_2, ..., q_c$;
$\tilde{g} = \sum_{j=1}^{c} w_j q_j$;
$x_{t+1} = x_t - \delta\tilde{g}$;

Algorithm 4: random quantileSGD (one iteration from local machine to master machine)

Input: learning rate δ, current point x_t,
Input: probability $p_1, ..., p_c$ s.t. $\sum p_j = 1$
$g_{t,i} = \nabla f_i(x_t)$ + sampling noise, i = 1, ..., M ;
from $\{g_{t,i}\}_{i=1}^{M}$, calculate c many candidate coordinate-wise quantiles: $q_1, q_2, ..., q_c$;
$\tilde{g} = q_j$ with probability p_j;
$x_{t+1} = x_t - \delta\tilde{g}$;

For example, if we have 5 machines and 5 gradient per iteration, we may calculate coordinately 25%, 50%(median), 75% quantiles of gradient and weight them by (0.25, 0.5, 0.25) or pick them with probability of (0.25, 0.5, 0.25); then use the weighted or randomly picked gradient for the GD on the master machine.

2.3 Discussions on quantile-SGD

These are some heuristic discussions about the algorithm.

- Making use of several 'central' quantiles ('central' means some quantiles percentage around the median) instead of only median can be viewed as 'smoothing' operation. The distribution of median may be asymmetric but smoothing contributes to symmetry. It may reduce the gap between median and mean.

- Calculating just a few coordinate-wise quantiles around median preserves similar robustness to median. Abnormal behavior of small proportion of local machine may not affect the gradient algorithm mach. It may be robust against the Byzantine failure.

The relationship between weighted quantileSGD and random quantileSGD is similar to GD and SGD. So we are going to check one of those two algorithm. The 'random quantile-SGD' will be implemented and its the performance will be compared with medianSGD and noisy medianSGD in section 3.

3 Experiments

3.1 Experiment settings and notations

The toy example $f(x) = \frac{1}{3}\sum f_i(x) = \sum(x - a_i)^2/2$, with $a_1 = 1, a_2 = 2, a_3 = 10$ in Chen et al. [2019] is not a good setting for our experiments.

The gap between median and mean are too large, so even the noisy SGD fails. If the magnitude of noise is too large, then gradient information is masked; otherwise, if the magnitude of noise is too small, then the gap can not be sufficiently reduced.

Therefore, our experiments consider synthetic 2-class multivariate Gaussian data and logistic regression model. Those simple settings are not either computationally intense or black-box, which is helpful to understand the medianSGD and noisy median SGD and check our algorithms.

3.1.1 Gaussian 2-class data

Generate training and testing data for two classes:

- $d = 5$, dimension of features is 5.
- For class 0 (negative class), random features $x \sim N(\mu_0, \Sigma)$, label $y = 0$
- For class 1 (positive class), random features $x \sim N(\mu_1, \Sigma)$, label $y = 1$
- $\mu_0 = (0, 0, 0, 0, 0)$, $\mu_1 = (1.5, 1.5, 1.5, 1.5, 1.5)$ are the mean vector of multivariate Gaussian.
- Σ is the covariance matrix of both classes. The diagonal entries of Σ is 1 and the off-diagonal entries are 0.5. Features have some correlation between each other.

The difference between μ_0 and μ_1 is not too far compared to the covariance, so the two classes will have some overlapped samples.

3.2 Heterogeneous distribution of data

Sample size:

- $N_{train} = 2500, N_{test} = 2000$: number of training samples is 2500; number of testing samples is 2000. They are independently generated under different random seeds (100) and (123).
- M = 5: there are 5 local machines and each of them has 500 training samples.
- The proportion of 0-label samples and 1-label samples are same for training data and testing data. But this ratio is not necessary to be $1 : 1$.

- The proportion of 0-label samples and 1-label samples can be very heterogeneous in each machine. For example, one machine can be full of 0-label samples and another machine may have 20% 0-label samples and 80% 1-label samples.

Machine #	node #1	node #2	node #3	node #4	node #5	Total
Number of 0-label sample	500	500	400	100	0	1500
Number of 1-label sample	0	0	100	400	500	1000

Table 1: The heterogeneous distribution of training data in each machine

- There are more 0-label training samples than 1-label training sample.
- The distribution of data inside each machine is also extremely unbalanced.
  - machine #1 and #2 only contain 0-label data; machine #3 has 80% 0-label samples.
  - machine #5 only contains 1-label data; machine #4 have 80% 1-label samples and 20%0-label samples;

3.2.1 Logistic regression

Solving logistic regression is a convex problem. Meanwhile, we can focus on the optimization problem itself because logistic regression is almost free of parameter tuning (except for the learning rate in GD).

For two-class logistic regression, linear combinations of the features are used to predict the log odds ratio ($\log \frac{p}{1-p}$) of one class:

$$\log \frac{p}{1-p} = w^T x \tag{2}$$

where parameters $w = (w_0, w_1, ..., w_d)^T$ is a $(d+1) \times 1$ vector including the intercept.

The loss function is in the form of cross-entropy (we consider mean instead of sum in object function to avoid additional issues of sample size):

$$w^* = \underset{w \in \mathbb{R}^{d+1}}{\operatorname{argmin}} L(w), \tag{3}$$

$$L(w) = -\frac{1}{N} \sum_{i=1}^{N} y_i \log p_i + (1 - y_i) \log(1 - p_i) \tag{4}$$

$$\text{where } p_i = \frac{1}{1 + e^{-w^T x_i}} \tag{5}$$

The gradient is a $(d+1) \times 1$ vector:

$$\nabla L(w) = -\frac{1}{N} \sum_{i=1}^{N} (y_i - p_i) x_i \tag{6}$$

$$\text{where } p_i = \frac{1}{1 + e^{-w^T x_i}} \tag{7}$$

Another benefit of logistic regression is that under Gaussian class generating mechanism st. $\Sigma_1 = \Sigma_2$, the estimation of p is the posterior probability (Bishop [2006]). In this sense, it is one

optimal model for our synthetic data. We do not need to worry about the representation power of logistic regression. The theoretical value of parameters are (when sample size goes to infinity, the 'exactly estimated' parameters will converge to theoretical values):

$$w_0 = -\frac{1}{2}\mu_1\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_0\Sigma^{-1}\mu_0 + \log\frac{\pi_1}{\pi_0} \tag{8}$$

$$(w_1, ..., w_d)^T = \Sigma^{-1}(\mu_1 - \mu_0) \tag{9}$$

where π_1 and π_0 are the population proportion of two classes. If two classes are balanced, $\pi_1 = \pi_0 = 1/2$

One remark is that even if we have learned the theoretical parameters, the classification accuracy cannot be 100%. Because two multivariate Gaussian clusters have overlaps and are not perfectly separable. A perfect separable model may overfit those synthetic training data.

Since we know the data-independent theoretical parameters' for logistic regression, we may compare our results to the theoretical parameters in section 3.

3.3 Algorithm details and hyper-parameters

6 algorithms are implemented to solve logistic regression:

1. SGD: average of gradients from local machines. This is the most straightforward algorithm but might be vulnerable to Byzantine failure.
2. medianSGD (Algo 1): take coordinate-wise median of gradients from local machines.
3. noisy medianSGD (Algo 2): add iid Gaussian noise before coordinate-wise median.
4. weighted quantileSGD (Algo3): calculate 25%, 50%, 75% coordinate-wise quantiles, and then average them with weights $(1/4, 1/2, 1/4)$
5. random quantileSGD 1 (Algo4): calculate 25%, 50%, 75% coordinate-wise quantiles, and then pick one of them with probability $(1/4, 1/2, 1/4)$
6. random quantileSGD 2 (Algo4): calculate 25%, 50%, 75% coordinate-wise quantiles, and then pick one of them with equal probability $(1/3, 1/3, 1/3)$. This a heavier smoothing operation compared to random quantileSGD 1.

Turning hyper-parameters:

- The batch size for each machine is 50. i.e. 50 samples are randomly picked to calculate gradient for every machine.
- Pick learning rate
  - $\delta = 0.01$ for first 3000 iterations;
  - After 3000 iterations, decrease δ to 0.005 for SGD and to 0.002 for the other 5 algorithms
- Noise magnitude $b = 0.5$. It is at a similar scale of the coordinate of each machine's gradient at final state.

3.4 Experiment results

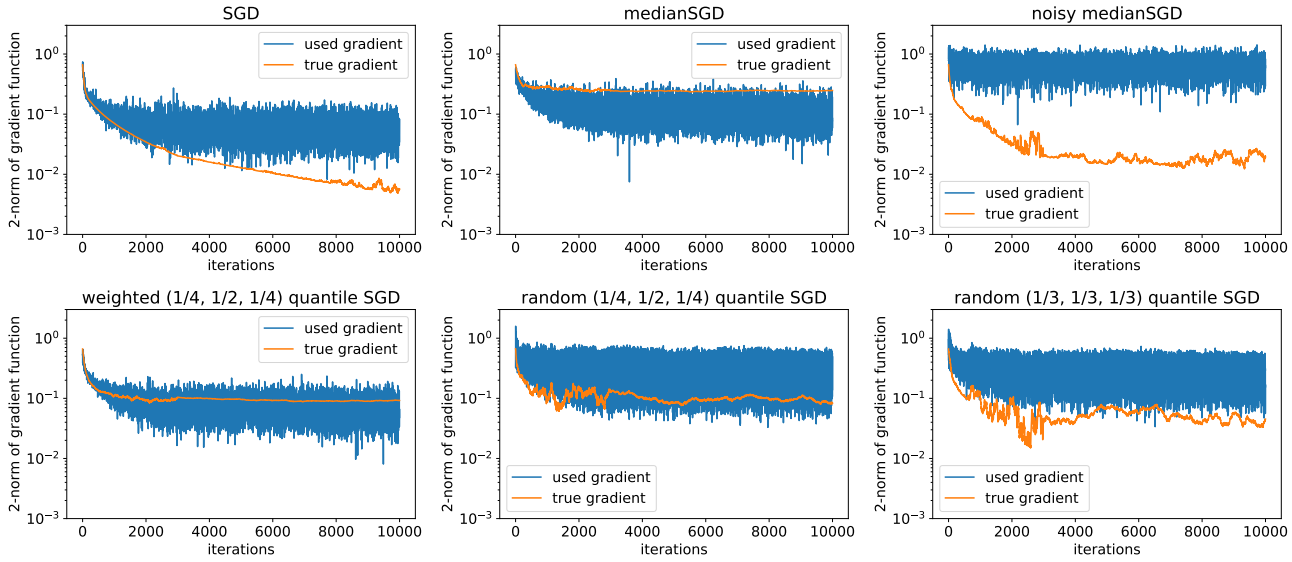


Figure 3: Norm of gradient for each algorithm (the scale of y-axis is fixed to be the same for 6 figures.)

Interpretations and observations from figure 3:

- The blue line 'used gradient' is the norm of the gradient directly used in GD procedure (for algorithm 3 and 4, it is the $\tilde{g}$). It can be mean of local gradients (SGD), or median of local gradients, or other form, which depends on the algorithm.
- The orange line 'true gradient' is the norm of gradient (averaged) over all training samples. This is the indicator of 'convergence'.
  - Standard SGD (meanSGD) convergence fastest.
  - The norm of median-SGD stucks above 10^{-1}. Also, we oberves the phenomenon in figure 1 (Chen et al. [2019]): even though the median gradient is small, the mean gradient may still be very large. Because the object is median gradient not the mean.
  - Noisy SGD behaves fairly good under proper noise magnitude. A bit inferior to standard SGD.
  - The behaviors of weighted quantile SGD and random quantile SGD are a bit inferior to noisy SGD but better than medianSGD. In particular, the random SGD with probability (1/3, 1/3, 1/3) is the best among 3 quantile SGD algorithms.

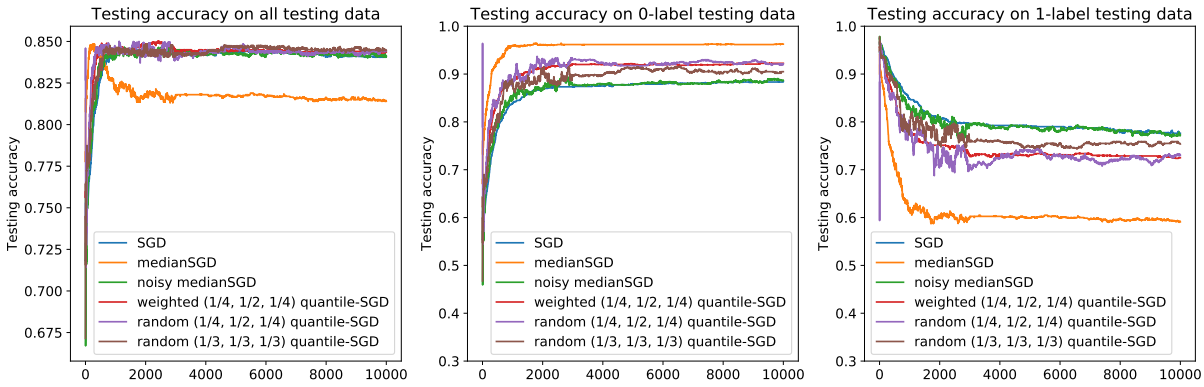


Figure 4: Testing accuracy on 1) whole testing set; 2) 0-label testing set; 3) 1-label testing set

Interpretations and observations from figure 4:

- Since two classes are unbalanced, we compare not only the prediction accuracy on whole testing data, but also the prediction accuracy for both classes. This helps us understand what happens for heterogeneous data.
- By the first plot in figure 4, it seems all algorithms work good on testing accuracy except for the medianSGD algorithm.
- The facts are more interesting in the second and third plot.
  - Even if artificial noise is added in noisy medianSGD, this algorithm has the closest classification accuracy on either class-1 and class-2 (blue and green lines are almost overlapped).
  - For the other methods, there is a favorable bias towards 0-label samples against 1-label samples. The medianSGD has the largest bias (the orange line): it has the highest testing accuracy rate on 0-label but the lowest testing accuracy on 1-label samples.
  - Even though our quantileSGD methods achieve very close testing accuracy to medianSGD and standard SGD, there is subtle difference in the final models. Since machine learning models can be *'good but different'* and we have already kept same proportion of each class in training and testing samples, we cannot assert that the bias is bad. But we need be aware of this *'subtle model preference'* and take it carefully in the practice.

	Intercept w_0	w_1	w_2	w_3	w_4	w_5
theoretical param [1]$(N_{train} \to \infty)$	-2.280	0.500	0.500	0.500	0.500	0.500
built-in func [2]	-2.320	0.449	0.465	0.490	0.384	0.698
SGD	-2.202	0.437	0.441	0.467	0.375	0.656
medianSGD	-2.993	0.680	0.280	0.369	0.324	0.645
noisy medianSGD	-2.162	0.374	0.451	0.448	0.440	0.585
weighted quantileSGD	-2.507	0.470	0.429	0.444	0.394	0.609
random quantile SGD 1	-2.527	0.484	0.437	0.448	0.396	0.621
random quantile SGD 2	-2.348	0.418	0.462	0.471	0.396	0.612

Table 2: The estimated parameters of logistic regression under different optimization algorithm

This table (2) shows similar messages from previous figures.

- The first row 'theoretical param' comes from equation (9), which only depends on the certain Gaussian data generating mechanism. It cannot be achieved through finite training samples.
- Since training sample size is limited, the solution from 'statsmodels' built-in function is not very close to theoretical values. The gap is reasonable.
- The parameters trained by medianSGD is the most different one. Many of the parameters bias much from the theoretical parameters, which implies that the algorithm may not converge well.
- Except for medianSGD, other algorithms achieve relatively similar parameters to built-in function. We cannot claim that noisy medianSGD learns parameters significantly better than quantile SGD.

By the above figures and table, we have seen the issues of medianSGD algorithm and checked the success of noisy medianSGD and the quantile-SGD algorithms. Some details help the understanding of those problems.

4 Conclusion and discussion

In this paper, with heterogeneous distributed synthetic data and logistic regression model, we check the failure of convergence of medianSGD and the success of noisy medianSGD. Our ideas: weighted quantile SGD and random quantile SGD algorithms also work good in this scenario. The quantile SGD may perform as one perturbation mechanism to solve the convergence issue on the heterogeneous data.

Some interesting phenomena are observed in the experiments, for example, even though testing accuracy is similar, models trained in different algorithms may have 'subtle preference' for one class (higher accuracy on one class). Extra attentions are needed when we are deal with practical distributed learning problems.

After the above analysis, **some questions of interests are still open:**

For the proposed quantile-SGD algorithm, it is still a very heuristic idea. Which quantiles should be picked around the median? What is the corresponding weights or probabilities for those quantiles? Several possible way to choose wights/probabilities: uniform distribution or symmetric double tail distribution.

To what extent of heterogeneity can quantileSGD still work for convergence? Since noisySGD has its limitation, for example, which may not work in the toy example in figure 1, the quantile-SGD algorithm might be easier to fail in that case. If we consider the 5 machines in our experiment, where 4 of them only contain 0-label samples, and 1 of them only catain 1-label samples, then the gradient information from 25% 50% 75% quantiles may still have a large gap to the mean.

Whether the quantileSGD works on more complex problem like deep neural network? Our experiments are only performed on special data and model, which are not realistic in practice.

To what extent that quantileSGD and noisy medianSGD robust against Byzantine failure? The Byzantine problem has not been explored in this project. Based on previous analysis, there is some robustness from quantile and there is still trade-off between robustness and convergence.

[1]The theoretical value in equation (9)

[2]Optimize over pooled data with functions in 'statsmodels' module. By default, the built-in function solves logistic regression by a variant of Newton's method from scipy, where Hessian matrix is usually replaced by Fisher information matrix. This method comes from statistical background.

References

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd: Compressed optimisation for non-convex problems, 2018.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Xiangyi Chen, Tiancong Chen, Haoran Sun, Zhiwei Steven Wu, and Mingyi Hong. Distributed training with heterogeneous data: Bridging median- and mean-based algorithms, 2019.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. Large scale distributed deep networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1223–1231. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf`.

Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982. ISSN 0164-0925. doi: 10.1145/357172.357176. URL `https://doi.org/10.1145/357172.357176`.

Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. *arXiv preprint arXiv:1803.01498*, 2018.