Assignment 2 of MATP6960: programming

(Due at 11:59PM on Oct-30-2021)

Instruction: Each student needs to submit the source code file and a report by Latex. **You will be evaluated based on whether you have all the results in the requirements below, and also the report**. Compress your source files (**DO NOT send the data.**) and PDF report file into a single .zip file, name it as "MATP6960_Assignment2_YourLastNameInitial", and send it to `optimization.rpi@gmail.com`

Problem description

In the class, we introduced the error-compensated stochastic gradient method; see the class note for the updates or page 49 of the survey paper [1]. In this project assignment, you are required to implement this method to train a neural network. Let $f_{\boldsymbol{\theta}}$ denote a neural network parameterized by $\boldsymbol{\theta}$. Then the training process will be to solve the following problem

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i), \tag{1}$$

where ℓ is a loss function such as the logarithmic softmax function in the provided code.

Requirements

Read the provided code `FashionMNIST.py`, which is modified based on the code for the first assignment. Again, you can use PyTorch to compute stochastic gradient but CANNOT use `optimizer.step()`. You are required to implement the error-compensated stochastic gradient method on a single machine. So **this is a simulation but NOT real distributed computing**. The following code creates data loaders, each of which will load the sampled data in a single class. The function `get_indices` has been provided. You can also write your own code to split the data set into subsets according to the labels. You can refer to any resource, including published papers/books and released code in GitHub, and you must give the reference in your report.

```
##############################################################
local_Xtrain = []
    local_ytrain = []

    for i in range(10):
        idx = get_indices(dataset_train, i)
        local_Xtrain.append(Xtrain[idx,])
        local_ytrain.append(ytrain[idx,])
##############################################################
```

You are required to do the follows.

1. Write a compress operator. This operator can be a quantization operator as we defined in class, or you can refer to Eqn. (3.1) in [1]. The compress operator can also be a sparsification operator, which picks a few largest elements (in magnitude) of the argument vector. Specifically, if you define a top-s sparsification operator Q_s, then for any given vector $\mathbf{v}$, $\mathbf{u} = Q_s(\mathbf{v})$ is the vector such that for each i,

$$u_i = v_i, \text{ if } v_i \text{ is one of the largest } s \text{ elements of } \mathbf{v}, \text{ and otherwise } u_i = 0.$$

2. Complete the `training` part in the `main` function in the provided code, or write your own `train` function by implementing the error-compensated stochastic gradient method. In the provided code, the batch size is set to 5 for each class of data, so for every update, the actual batch size would be 50, because each update uses an aggregated gradient. You can tune the batch size.

3. Pick either the fully-connected network or the LeNet, or you can define another architecture. Test your `train` function by using your defined compress operator and running the method to at least 50 epochs or until you see no progress based on the testing accuracy. If you use a b-bit quantization operator, you are required to try different values of b, say $b = 4, 8, 16, 32$. If you use a top-s sparsification operator, you also need to try different values of s, for example $s = 0.1n, 0.2n, 0.5n$, where n is the dimension of the model parameter. Report the results for different values of b or s. You may need to use different learning rates for different b or s.

4. Discuss what you observe in the test. For example, how you tune the algorithm parameters, how the values of b or s affect the prediction accuracy of the algorithm, how the batch size affects the stability of the algorithm. If you use a big b or s, say, $b = 32$, or $s = 0.8n$, the testing accuracy should be near 89%, just as that by a non-compressed method.

Bonus (up to 20% bonus credit)

If you use both quantization and sparsification operators, you will earn up to 20% bonus credit.

References

[1] J. Liu, C. Zhang, et al. Distributed learning systems with first-order methods. *Foundations and Trends® in Databases*, 9(1):1–100, 2020.