Low Rank Communication for Federated Learning

Huachi Zhou(✉), Junhong Cheng, Xiangfeng Wang, and Bo Jin

East China Normal University, Shanghai 200062, People's Republic of China
{hczhou,jhcheng}@stu.ecnu.edu.cn,
{xfwang,bjin}@cs.ecnu.edu.cn

Abstract. Federated learning (FL) aims to learn a model with privacy protection through a distributed scheme over many clients. In FL, an important problem is to reduce the transmission quantity between clients and parameter server during gradient uploading. Because FL environment is not stable and requires enough client responses to be collected within a certain period of time, traditional model compression practices are not entirely suitable for FL setting. For instance, both design of the low-rank filter and the algorithm used to pursue sparse neural network generally need to perform more training rounds locally to ensure that the accuracy of model is not excessively lost. To breakthrough transmission bottleneck, we propose low rank communication Fedlr to compress whole neural network in clients reporting phase. Our innovation is to propose the concept of optimal compression rate. In addition, two measures are introduced to make up accuracy loss caused by truncation: training low rank parameter matrix and using iterative averaging. The algorithm is verified by experimental evaluation on public datasets. In particular, CNN model parameters training on the MNIST dataset can be compressed 32 times and lose only 2% of accuracy.

Keywords: Federated learning · Convolutional neural network · Low rank approximation · Matrix compression · Singluar vaue decomposition

1 Introduction

The widespread application of deep neural networks has achieved remarkable success in many computer tasks, such as image processing, speech recognition, and text translation. However, most of them require people to share their personal data with service providers, including their daily behaviors, personal preferences, etc., which to some extent is quite private data. Therefore, users hate or even refuse to upload personal data to the server.

At the same time, mobile phones and tablets with a large amount of users' sensitive data become more and more powerful. With development of edge computing, the scheme of deploying tasks at the edge of the network rather than the cloud is becoming more and more mature.

Y. Nah et al. (Eds.): DASFAA 2020 Workshops, LNCS 12115, pp. 1–16, 2020.
https://doi.org/10.1007/978-3-030-59413-8_1

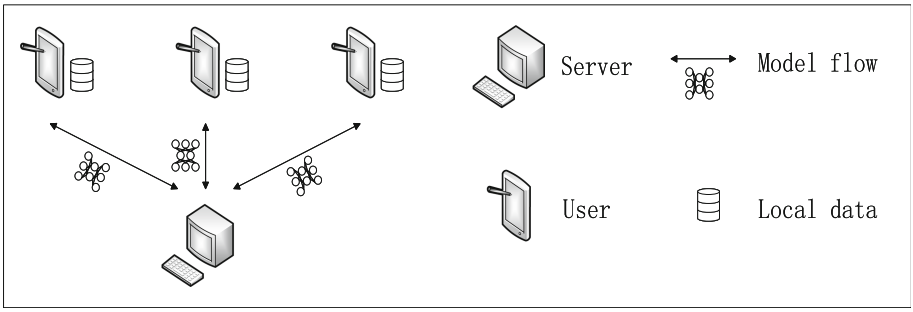

Fig. 1. The structure of FL

FL is a machine learning framework that aims to build a high-quality learning model as centralized in which training data remain local. In order to do this, H. Brendan et al [1] propose Federated Average algorithm. Terminal devices train a shared model with data stored locally under the coordination of parameter server. At each training round, sever randomly selects those volunteering devices which are qualified to participate in this round. Those chosen clients upload their own gradients to the central sever within a fixed time. Then server averages all participants' gradients and sends it back to the participants. This iterative training process continues throughout the network until global convergence is reached or some termination condition is met [1]. However, in the learning process of mass clients, communication efficiency is of the utmost importance (Fig. 1).

On one hand, because of characteristics of asymmetric digital subscriber line, the upload speed is several times lower than the download speed. Network communication speeds are even orders of magnitude slower than distributed environment. In order to solve transmission bottleneck, reducing communication costs, especially when clients upload their models, helps nodes spend less time on crowded channels to speed up training round.

On the other hand, existing work trying to cut communication cost mostly focuses on two schemes. The first is to reduce the amount of information needed to be exchanged globally by increasing the number of local iteration rounds [2]. However, the size of model parameters needed to be transmitted for a single time is still very large. The second is to quantify information to reduce the number of bits [3,4]. But, the problem of quantization is that the accuracy is unstable and it is easy to cause precision loss.

To solve these problems, we propose Federated low rank(Fedlr) algorithm which reduces the communication cost between server and clients, while keeping the accuracy. Our work shows that the low-rank representation of a matrix can optimally achieve a balance in accuracy loss, which brings new ideas to the design of communication efficiency strategy. Our contributions are summarized as follows:

– We apply truncated SVD to compress parameter matrix of model convolutional layers and propose a general algorithm for determining the optimal compression ratio of the parameter matrix.
– In order not to significantly increase model loss caused by rank truncation, we train low rank convolutional layers. The central node aggregates historical gradients of each node separately to reduce the variance of current gradient.
– The extensive experiments on public datasets show the effectiveness of the proposed algorithm.

2 Related Works

In Sects. 2.1 and 2.2, we introduce traditional neural network compression techniques. In Sect. 2.3, we introduce gradient compression techniques. In our description, readers will find that content in Sect. 2.3 is more suitable for FL.

2.1 Model Pruning

The purpose of neural network pruning is to reduce the amount of model weights to make model thinner. There are three distinguished mechanisms. The first is to add different penalty terms to the loss function of neural network during training process to promote the trained neural network to contain more zero elements. [5] selects parameter index set whose value range is {0,1} to penalize the total number of network parameters. [6] sets a scaling factor which multiplies with a channel output before it goes into next layer. The penalty term chooses the sum of scaling factors. The second method directly deletes smaller weights during training. [7] proposes a gentle gradual pruning method, which slowly prunes from a small sparsity to the desired sparsity. [8] inherits this idea and proposes one-shot pruning, while each round masking some smaller weights, resetting remaining weights to initial value and retraining the whole network. In the forward propagation process, [9] deletes a channel of the i-th layer and see if the output of the i+1 layer is seriously affected to determine the importance of the channel weight in i-th layer. The third method is weight sharing: through weight clustering, channels of the same category share the same weight [10], which only decreases the memory space occupied by the model and does not save model reasoning time, compared to the first two methods. Overall, pruning is not suitable for FL environments. First, it is difficult to agree on whether to cut off some connectivity between neurons among clients. Second, pruning needs extra rounds for retraining to restore the performance of the model, which takes a lot of time.

2.2 Model Low Rank Filter and Model Quantization

Another method of model compression is to use low rank convolutional filters. [11] designs low rank filters including two processes. First, it classifies input

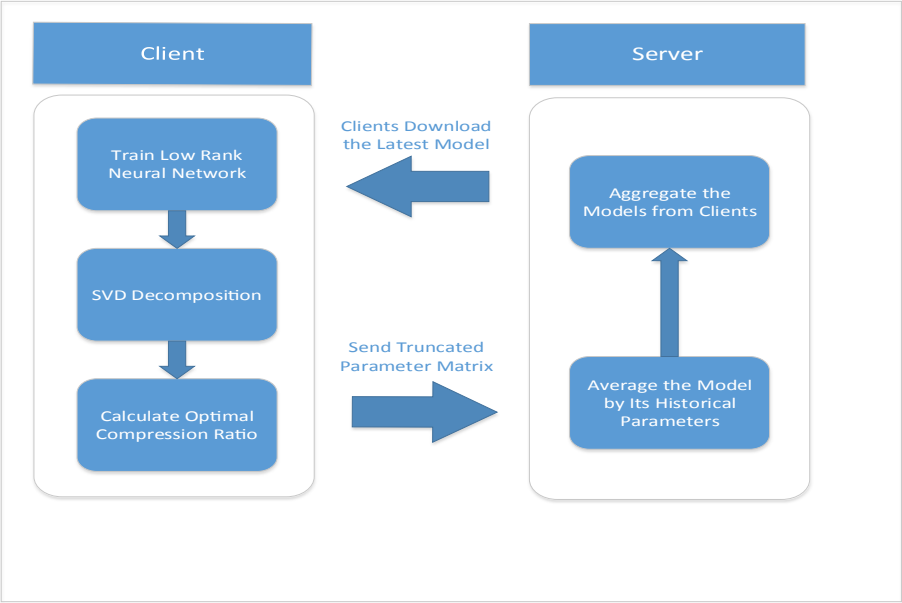

Fig. 2. Flow chart of Fedlr algorithm

channels and only allows portion of filters to connect to a specific class to reduce filters redundancy and projects the input image dimension down to 1D subspace using intermediate filters to reduce input size. Then they apply the tensor decomposition method to all filters to store weights more efficiently. [12] In order to reduce channel or filter redundancy, horizontal and vertical filters with a rank of 1 are used to approximate the previous 2d tensor. The aim of model quantization is to replace floating-point numbers with 8-bit or 16-bit integers or other low-precision numeric formats. Based on [13], there are two quantization methods: deterministic quantization and stochastic quantization. For the former, the popular binary neural network maps model weights to $\{+1, -1\}$ according to their positive and negative. For the latter, [14] assumes model weights follow multinomial distribution and attempts to train neural networks with discrete weights. But under FL setting, low rank filter design also needs additional local training rounds to minimize reconstruction errors, which takes a lot of time. For quantization, decreasing number of bits may cause serious degradation of the model. According to Deep Compression's survey, quantized convolutional layer requires 10 bits to avoid a significant loss of accuracy [15].

2.3 Gradient Quantization

Gradient quantization, just as its name suggests, is applied to the output gradient to be transmitted by each node, which follows a common principle. The mathematical expectation of quantified variable is an unbiased estimate of the

original variable mathematical expectation [16–18]. Unlike model quantization, which recovers the accuracy of pruned model by increasing training round, gradient quantization uses other strategies to bridge the gap between quantized values and true values of gradients. [19] Each node accumulates its own quantization error for each round. Before each round starts quantization, decayed quantization error is added to current local gradient to compensate it. [20] uses stochastic rotated quantization to limit quantization range to $[0, 1]$ to reduce mean squared error. Due to its wide application in distributed learning, gradient quantization is also widely used in FL. It is worth noting that our work does not include any distributed quantization and the subsampling techniques contained in [21]. Future work will try to make them compatible with our work to provide greater compression rates (Fig. 3).

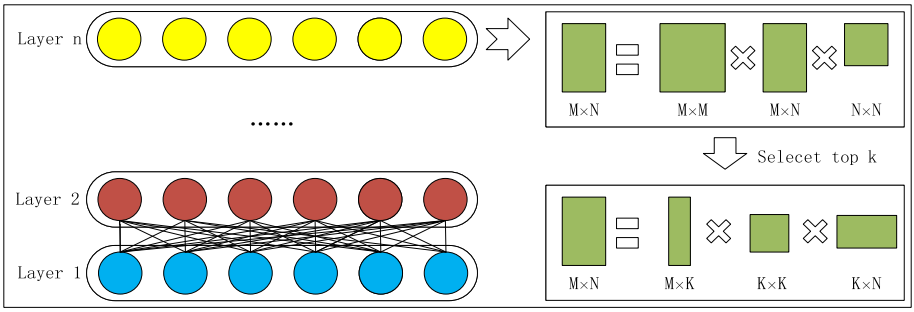

Fig. 3. The left part is n fully connected layers and we assume top layer has shape $M \times N$. Right part illustrates the compression process of SVD

3 Federated Low Rank Algorithm

3.1 Overview

In this section, we will describe our method. The algorithm has four main components: training low rank parameter matrix, determining optimal compression ratio, compressing model using SVD and reducing noise. The complete procedures at the server and each client node are presented in Algorithms 1 and 2, respectively. In order to explain our Fedlr strategy more clearly, we provide a flow chart (shown in Fig. 2) of the algorithm. In left part, clients download latest model, train low rank model locally and upload their model using truncated SVD. In right part, server receives each client model, averages with its historical model and aggregates all averaged models.

Algorithm 1: Fedlr strategy for server:

Input : specified minimum receiving client gradient number T, Specified communication round N, Model selected for training $\widetilde{W}$, volunteering edge nodes set C

Output: shared model parameter W_{out}

Initialize $\widetilde{W}$ randomly ;
Initialize $history\_gradients \leftarrow []$;
Initialize $aggregated\_gradients \leftarrow []$;
for *each communication round r = 1,2... N* **do**
    Randomly select S_t clients from C;
    send $\widetilde{W}$ to S_t clients;
    for $t = 1, 2...T$ **do**
        listening clients m in S_t and receiving message $\widetilde{U_m}\widetilde{\Sigma_m}\widetilde{V_m}$;
        $W_m = \widetilde{U_m}\widetilde{\Sigma_m}\widetilde{V_m}$;
        history_gradients[m].append (W_m) ;
        $\widetilde{W_m} = \sum_{i=1}^{3} history\_gradients[m][i]$;
        aggregated_gradients.append($\widetilde{W_m}$);
    end
    $\widetilde{W} = \Sigma_{i=1}^{T} aggregated\_gradients[i]$;
end
return $\widetilde{W}$ as W_{out};

3.2 Model Compression Using SVD

Low-rank decomposition has a wide range of applications in image restoration, matrix filling, and collaborative filtering. Projecting matrices to lower-dimensional linear subspaces greatly reduces costs of federal learning communication [22]. The adversarial training of Peter Langenberg and others on the fully-convolutional neural network shows that low-rankness of neural network weights can further improve robustness [23]. Low rank characteristic also helps model from overfitting. Compared to other popular matrix factorization techniques, such as CUR matrix factorization, which maintains sparsity within the decomposed matrix, we adopt a more popular truncated SVD matrix technique. There are two advantages. First, according to Eckart-Young theorem, truncated SVD has the smallest low-rank approximation error, which lets us not worry about excessive decline in accuracy. Second, according to extensive experiments, only around 1% of singular values contain 97% of energy, which also explains from another perspective why the SVD low rank approximation still maintains excellent accuracy even if a large number of singular vectors are discarded. This feature of focusing energy on a few singular values allows us to compress the parameter matrix as much as possible. In this paper, we only focus on parameter matrix of convolutional layer and do not consider structure or weight of a single filter in particular. In results analysis section, we will discuss connection between our strategy and single convolution filter. The weight of CNN layer

Algorithm 2: Fedlr strategy for clients:

Input : shared model parameter from server W, local datasets D, minibatchsize b, learning rate r, epoch size T, hyperparameter λ α, loss function l, Frobenius norm constraint F, clients C

Output: model parameter $\widetilde{U}\widetilde{\Sigma}\widetilde{V}$

```
training phase;
for each client c ∈ C in parallel do
    Initialize W_{n,c} ← W;
    for m = 1,2...T do
        for n = 1,2...D/b do
            W_{n,c} = W_{n-1,c} − r(∇l + α∇F);
        end
    end
    Σ_{i=1}^{r} UΣV = SVD(W_c);
    Estimate k ← optimal_ratio(Σ) from equation (14);
    Sending Ũ Σ̃ Ṽ to server;
end
```

is in the form of 4d tensor $W^{d\times d\times m\times n}$(d is kernel size, m is the number of input channel and n is the number of output feature map). We convert W into $d^2 \times m \times n$ shape. So each element in 3d array is $W \in R^{m\times n}$. According to the definition of SVD [24], $W = U\Sigma V$, where U is a left singular matrix with the size of $m \times \min(m, n)$, V is a right singular matrix with the size $\min(m, n) \times n$ and Σ is a diagonal matrix of singular values like $diag(\sigma_1, \sigma_2.....,\sigma_n)$ where $\sigma_1 \geq \sigma_2..... \geq \sigma_{\min(m,n)}$. According to the definition of SVD, W can be written as

$$W = \sum_{i=1}^{\min(m,n)} \sigma_i U_i V_i \ . \tag{1}$$

Assuming that only k ranks are retained, the optimal rank k approximation of the parameter matrix W is:

$$\widetilde{W} = \sum_{i=1}^{k} \sigma_i U_i V_i \ . \tag{2}$$

The size of original parameter matrix $B = m \times n$. After the parameter matrix is compressed, message sent by a client to the server equals to $\widetilde{B}$ (We don't count the packet header and tail added for network transmission):

$$\widetilde{B} = \widetilde{U}\widetilde{\Sigma}\widetilde{V} = m \times k + k + n \times k = k(m + n + 1) \ . \tag{3}$$

In order to achieve message compression, the reserved rank number k needs to satisfy $0 \leq \widetilde{B} \leq B$, which means

$$0 \leq k \leq \frac{mn}{m + n + 1} \ . \tag{4}$$

We operate each element in 3d array of all convolutional layers in same way. When the channel is congested and packet loss occurs frequently, mobile phone detects that network layer is in a bad condition with high delay. Instead of sending entire low rank matrix at once, the client sends one singular value and the corresponding left and right singular vector each time.

3.3 Optimal Compression Ratio

In order to reduce redundancy between filters, it is really effective to replace them with a linear combination of fewer filters or low-rank filters. In practical applications, it is very difficult to decide how many filters to be left in order to achieve the maximum compression ratio while a small performance degradation. Researchers face the same problem when compressing neural parameters to be transmitted. In the past, model compression often selects fixed-rank interception [25]. We supppose that different machine learning tasks and corresponding datasets demand different parameter space. The result is that the number of ranks worth retaining for different task or different layer is also different. The results analysis section verifies our assumption. When fixed compression ratio is large, information originally contained in the parameter matrix is excessively deleted. When the compression ratio is too small, the potential for deeper compression of a matrix is ignored. Considering this problem, we propose a formula to decide optimal compression rate of a parameter matrix. Suppose $g(k)$ represents the compression ratio of a parameter matrix, and $f(k)$ represents the loss rate of a matrix information. Based on Sect. 3.1, the optimal k needs to satisfy

$$\max_k \ f(k) + \alpha g(k) \ , \quad where \ \ 0 \leq k \leq \frac{mn}{m+n+1}. \tag{5}$$

We use Frobenius norm to measure the retention rate of matrix information:

$$f(k) = \frac{\left\|\widetilde{W}\right\|_F}{\|W\|_F} \ , \tag{6}$$

and rank compression rate to measure matrix compression rate:

$$g(k) = \frac{r-k}{r} \ . \tag{7}$$

Now (5) becomes to

$$\max_k \frac{\left\|\widetilde{W}\right\|_F^2}{\|W\|_F^2} + \left(\frac{r-k}{r}\right)^2 \ . \tag{8}$$

Based on definition of Frobenius norm,

$$\|W\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2} = \sqrt{trace(W \times W)} = \sqrt{\sum_{i=1}^{\min(m,n)} \sigma_i^2} \ , \tag{9}$$

and

$$\left\|\widetilde{W}\right\|_F = \sqrt{\sum_{i=1}^{k} \sigma_i^2} \ . \tag{10}$$

For the purpose of optimizing this formula more easily, we square $f(k)$ and $g(k)$. So we get

$$\max_k \frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{\min(m,n)} \sigma_i^2} + \alpha(\frac{r-k}{r})^2 \ . \tag{11}$$

We set $f(k)$ first-order increment symbol to

$$\nabla f(k) = \frac{\sigma_k^2}{\sum_{i=1}^{\min(m,n)} \sigma_i^2} \ , \tag{12}$$

first-order increment of $g(k)$

$$\nabla g(k) = \frac{\alpha(2k-2r-1)}{r^2} \ . \tag{13}$$

We can easily see that $f(k)$ is a monotonically increasing function, and the increment is gradually decreasing. $g(k)$ is a monotonically decreasing function and absolute value of the increment is gradually decreasing. Judging from the second-order increment of $f(k)$ and $g(k)$, the change rate of first-order increment of $f(k)$ goes from large to small and the change rate of first-order increment of $g(k)$ is constant. And because front singular values concentrate most of energy, when k is small, the first-order increment of $f(k)$ should be greater than the first-order increment of $g(k)$. Moreover, we assume that k should not reach the optimal value near the upper and lower bounds of the domain. So, when the first-order increment of $f(k)$ is equal to the first-order increment of $g(k)$, the whole formula reaches maximum. So optimal k satisfies

$$\alpha(2r+1) \sum_{i=1}^{\min(m,n)} \sigma_i^2 = r^2\sigma_k^2 + 2\alpha \sum_{i=1}^{\min(m,n)} \sigma_i^2 k \ . \tag{14}$$

Among them, α is a hyperparameter, which measures the importance of matrix compression ratio. When the channel is extremely congested and packet loss occurs seriously, clients increase α value appropriately. When network connection is smooth, clients decrease α value. More importantly, it is worth noting that parameters needed for getting k totally depend on singular values of matrix $W \in R_{m\times n}$ mentioned in Sect. 3.1 and do not require additional calculations to occupy computing time of edge nodes. So it does not bring burden to hardware devices and saves cell phone battery power.

3.4 Train Low Rank Parameter Matrix

During truncation, although redundancy and unimportant information are deleted, it is still harmful to neural network powerful expression ability. We

hope to minimize the accuracy loss caused by the truncation of singular values. So we train neural network parameters into a rank compact matrix whose energy is more focused on previous singular values. The benefit of the parameter matrix to be low rank for neural network is twofold. First, it brings greater generalization performance. Second, truncation of decomposed matrix sent in the reporting stage will not cause too much loss of accuracy. Generally, we assume neural network loss function l. So our training task is:

$$\min l(x; w) \ . \tag{15}$$

In order to ensure that the trained matrix is a low-rank matrix, we add the nuclear norm to constrain the parameter matrix W. Now our task is

$$\min l(x; w) + \lambda \left\| W \right\|_* \ . \tag{16}$$

The hyperparameter λ measures the influence of nuclear norm on the entire formula. One strategy is to use subgradient descent to solve this problem. First $W = U\Sigma V$ is SVD of W, then the sub-gradient of the above formula is $\nabla l + UV^T$. The sub-gradient method can be used for non-differentiable objective functions and can be applied to a wider range of problems. However, sub-gradient method is much slower than stochastic gradient descent method, which will cause the problem of slow convergence. Another way is to use proximity operator [26]:

$$W_{k+1} = prox_{\gamma\lambda\|\Delta\|_*}(W_k - \gamma\nabla l(W_k)) \tag{17}$$

to solve the problem. Later, it applies soft-thresholding operator to W. However, it still involves singular value decomposition of the parameter matrix W in each mini batch gradient descent round, which is not time-efficient. So we replace the nuclear norm with Frobrenius norm which has the advantages of being smooth and differentiable to constrain parameter matrix rank. Now our task is:

$$\min l(x; w) + \lambda \left\| W \right\|_F \ . \tag{18}$$

3.5 Noise Reduction Method

Because common machine cannot store gradients of all training samples in its memory, SGD feeds a fraction of samples to models to complete this training round. The direction of the gradient descent completely depends on gradient calculation results of current mini batch, resulting in a large variance. When model gets to around local optimal point, the noise will make it oscillate back and forth around destination and cannot shrink immediately. We treat low-rank truncation of model weight matrix as a kind of noise applied to the gradient. According to Jorge Nocedal et al, there are three ways to reduce variance [27]. The first type is the dynamic sampling method, which reduces the variance of gradient estimation by gradually increasing sample size when calculating gradient. The second type is iterative averaging method which averages the parameter matrix $\widetilde{W}$ obtained after each training round with historically stored parameter matrix

to reduce its variance; The third type is gradient aggregation method, which stores historical gradient of each sample and directly or indirectly uses historical or current gradients of all samples to make corrections during each mini-batch estimation of gradient. Compared to the second method, both first and third methods require edge nodes to store many gradients locally. We are more willing to transfer this workload to the parameter server which performs only gradient aggregation task before. Let the server store the historical parameter matrix sent by each client at each round. The method is described as follows. Before server applying weight aggregation, the parameter server first uses the tail averaging method [28]:

$$\widetilde{w_{k+1}} \leftarrow \frac{1}{k-s+1}\Sigma_{j=s+1}^{k+1}\widetilde{w_j} \tag{19}$$

to reduce its variance for each client. Compared to complicated explanations given in [28], it is very easy to understand why (20) only averages the nearest $k-s+1$ historical gradients. According to the definition of model's expected generalization error:

$$E_{model}(f;D) = E_D[(f(x;D)-y)^2] = E_D[(\overline{f}(x)-y)^2] + E_D[(f(x;D)-\overline{f}(x))^2], \tag{20}$$

the initial model has a poor ability to fit the data set, so its variance is large and the bias is small. As the number of training rounds increases, the model becomes more complex, characterized by small bias and large variances. Only recent models can help maintain a low-bias feature while reducing variances.

Table 1. Model

3×3 conv. 96 ReLU
3×3 conv. 96 ReLU
3×3 conv. 96 ReLU stride 2
3×3 conv. 192 ReLU
3×3 conv. 192 ReLU
3×3 conv. 192 ReLU stride 2
3×3 conv. 192 ReLU
1×1 conv. 192 ReLU
1×1 conv. 10 ReLU
Global averaging layer
10 or 100-way softmax

4 Experiment

4.1 Datasets

We evaluate the performance of our algorithm on 3 datasets: MNIST [29], CIFAR-10 and CIFAR-100 [30]. MNIST is a large handwritten digit datasets

with a training set of 60,000 examples and a test set of 10,000 examples. The CIFAR-10 data set consists of 10 types of 32×32 color pictures, which contains a total of 60,000 pictures, and each type contains 6000 pictures. 50000 pictures are used as the training set, and 10,000 pictures are used as the test set. The CIFAR-100 dataset has 100 categories, and the number of pictures in each category is one tenth of CIFAR-10.

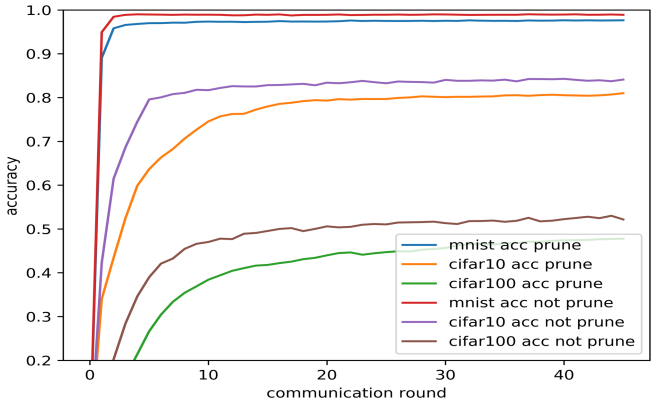

Fig. 4. It shows changes in accuracy on test set and convergence of three datasets after adopting our strategy. Not prune means not adopted.

4.2 Parameter Setting

For us, training a high accuracy model is not our goal. We only care about whether the accuracy can be maintained while hugely improving communication efficiency after using our Fedlr strategy. The learning rate, decay rate, momentum, α, λ is set to 0.01, 10^{-6}, 0.9, 1, 0.01. For Sect. 3.4, sever keeps each client 3 historical gradients. We set 5 clients, each of which gets 10,000 pictures after training dataset has been shuffled. The aggregation algorithm we adopt is Mcmahan federated averaging algorithm [1]. The epoch size of each client training round is 10 and the communication round is 45.

4.3 Model

We use All-CNN-C model (shown in Table 1) in [31] which implementes a convolutional neural network with all convolutional layers. More convolutional layers mean that we have more chances to compress our model, which makes our strategy effects more convincing. This CNN model has a total of eleven layers. The first seven layers are in the form of 4d tensor and can be processed by our strategy. The other layers have much fewer parameters than them. So they are not worth being compressed by our method.

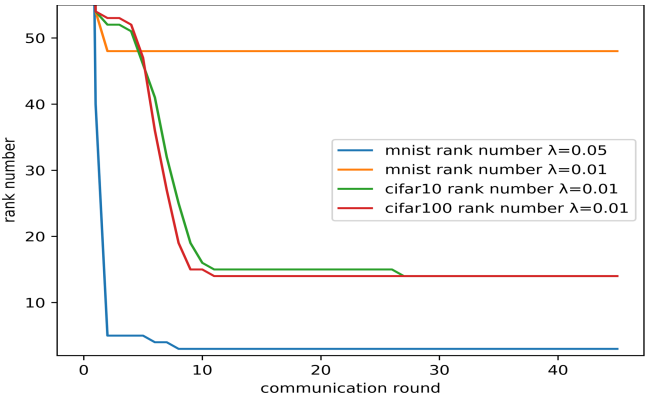

Fig. 5. We show the effect of our strategy by average optimal k drop in the model fifth layer. Initial rank is 192.

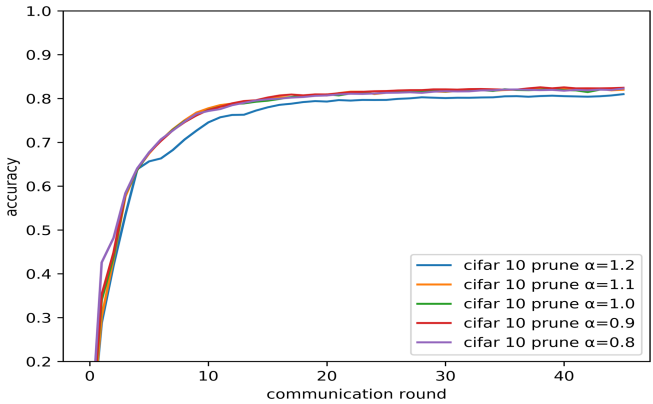

Fig. 6. We show CIFAR-10 datasets test accracy among different α in Sect. 3.2 to see the effect of hyperparameters on the final accuracy of our model.

4.4 Results Analysis

As shown in Fig. 4, the experimental results show that our Fedlr strategy performs very well on three datasets MNIST, CIFAR-10 and CIFAR-100. On MNIST datasets, by sacrificing only 2% accuracy, optimal rank k reduces from 192 to 3 (Fig. 5), achieving 64× parameter matrix rank compression rate. In fact, because the left singular matrix and the right singular matrix are transmitted at the same time, model compression rate is actually only half of rank compression rate, 32×. And model only loses 4% accuracy on CIFAR-10 datasets and convergence rate does not change much compared with normal FL without any compression strategy. Although rank compression rate in the first few rounds is only 4× for three datasets, it improves very fast. Only in a dozen rounds optimal rank k could converge to a fixed value. In hundreds of rounds of FL, these times

can be omitted. At the same time, we notice the abnormal performance of rank descent on the MNIST dataset. We think that CNN is very easy to learn the characteristics of digital pictures. So model convergence rate is very fast. The regularization term does not bring constraints to model weights. We need to adjust hyperparameter λ in Sect. 3.3 from 0.01 to 0.05. The result proves our opinion. The CIFAR datasets do not have such problems, so we do not make this adjustment for them. As shown in Fig. 6, We find that final accuracy of the model is not sensitive to the hyperparameter α. We think there are two possible reasons. The first point is that the number of clients is small and owing training set is large, they are very easy to learn core knowledge. The second point is that we train a low-rank parameter matrix, which causes the previous singular values to be large. It is difficult to decrease the optimal k even with a larger hyperparameter α. These two reasons inspire our future work to explore the correlation between the value of α and the accuracy of the model when there are a large number of clients with a small training set. We hope this research will provide a minimum bound for the number of needed filters to reconstruct original convolutional layers and guide the design of low-rank filters.

5 Conclusions

Under FL setting, we use a low-rank communication to compress model parameter matrix. In a communication network, parameter server can quickly receive clients responses without worrying about the obvious loss of accuracy. Our algorithm is verified by extensive experimental evaluation on public datasets.

Acknowledgement. This work was supported by National Key R&D Program of China (No. 2017YFC0803700), NSFC grants (No. 61532021 and 61972155), Shanghai Knowledge Service Platform Project (No. ZF1213) and Zhejiang Lab (No. 2019KB0AB04).

References

1. McMahan, H.B., Moore, E., Ramage, D., Hampson, S.: Communication-efficient learning of deep networks from decentralized data. arXiv preprint arXiv:1602.05629 (2016)
2. Liu, Y., et al.: A communication efficient vertical federated learning framework. arXiv preprint arXiv:1912.11187 (2019)
3. Reisizadeh, A., Mokhtari, A., Hassani, H., Jadbabaie, A., Pedarsani, R.: FedPAQ: a communication-efficient federated learning method with periodic averaging and quantization. arXiv preprint arXiv:1909.13014 (2019)
4. Sattler, F., Wiedemann, S., Müller, K.R., et al.: Robust and communication-efficient federated learning from non-IID data. arXiv preprint arXiv:1903.02891 (2019)
5. Srinivas, S., Subramanya, A., Venkatesh Babu, R.: Training sparse neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 138–145 (2017)

6. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2736–2744 (2017)
7. Zhu, M., Gupta, S.: To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv preprint arXiv:1710.01878 (2017)
8. Frankle, J., Carbin, M.: The lottery ticket hypothesis: finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635 (2018)
9. Luo, J.H., Wu, J., Lin, W.: Thinet: a filter level pruning method for deep neural network compression. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 5058–5066 (2017)
10. Ullrich, K., Meeds, E., Welling, M.: Soft weight-sharing for neural network compression. arXiv preprint arXiv:1702.04008 (2017)
11. Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in Neural Information Processing Systems, pp. 1269–1277 (2014)
12. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. arXiv preprint arXiv:1405.3866 (2014)
13. Guo, Y.: A survey on methods and theories of quantized neural networks. arXiv preprint arXiv:1808.04752 (2018)
14. Shayer, O., Levi, D., Fetaya, E.: Learning discrete weights using the local reparameterization trick. arXiv preprint arXiv:1710.07739 (2017)
15. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015)
16. Konečný, J., Richtárik, P.: Randomized distributed mean estimation: accuracy vs. communication. Front. Appl. Math. Stat. **4**, 62 (2018)
17. Alistarh, D., Grubic, D., Li, J., Tomioka, R., Vojnovic, M.: QSGD: communication-efficient SGD via gradient quantization and encoding. In: Advances in Neural Information Processing Systems, pp. 1709–1720 (2017)
18. Horvath, S., Ho, C.Y., Horvath, L., Sahu, A.N., Canini, M., Richtarik, P.: Natural compression for distributed deep learning. arXiv preprint arXiv:1905.10988 (2019)
19. Wu, J., Huang, W., Huang, J., Zhang, T.: Error compensated quantized SGD and its applications to large-scale distributed optimization. arXiv preprint arXiv:1806.08054 (2018)
20. Suresh, A.T., Yu, F.X., Kumar, S., McMahan, H.B.: Distributed mean estimation with limited communication. In: Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 3329–3337. JMLR. org (2017)
21. Caldas, S., Konečny, J., McMahan, H.B., Talwalkar, A.: Expanding the reach of federated learning by reducing client resource requirements. arXiv preprint arXiv:1812.07210 (2018)
22. Prabhavalkar, R., Alsharif, O., Bruguier, A., McGraw, L.: On the compression of recurrent neural networks with an application to LVCSR acoustic modeling for embedded speech recognition. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5970–5974. IEEE (2016)
23. Langeberg, P., Balda, E.R., Behboodi, A., Mathar, R.: On the effect of low-rank weights on adversarial robustness of neural networks. arXiv preprint arXiv:1901.10371 (2019)
24. Kalman, D.: A singularly valuable decomposition: the SVD of a matrix. Coll. Math. J. **27**(1), 2–23 (1996)

25. Konečný, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)
26. Ciliberto, C., Stamos, D., Pontil, M.: Reexamining low rank matrix factorization for trace norm regularization. arXiv preprint arXiv:1706.08934 (2017)
27. Bottou, L., Curtis, F.E., Nocedal, J.: Optimization methods for large-scale machine learning. SIAM Rev. **60**(2), 223–311 (2018)
28. Jain, P., Kakade, S.M., Kidambi, R., Netrapalli, P., Sidford, A.: Parallelizing stochastic approximation through mini-batching and tail-averaging. STAT **1050**, 12 (2016)
29. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
30. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images (2009)
31. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: the all convolutional net. arXiv preprint arXiv:1412.6806 (2014)