Optimized Quantization for Convolutional Deep Neural Networks in Federated Learning

You Jun Kim, Choong Seon Hong
Department of Computer Science and Engineering
Kyung Hee University, 17104
Republic of Korea
{yj4889, cshong}@khu.ac.kr

Abstract—Federated learning is a distributed learning method that trains a deep network on user devices without collecting data from central server. It is useful when the central server can't collect data. However, the absence of data on central server means that deep network compression using data is not possible. Deep network compression is very important because it enables inference even on device with low capacity. In this paper, we proposed a new quantization method that significantly reduces FPROPS(floating-point operations per second) in deep networks without leaking user data in federated learning. Quantization parameters are trained by general learning loss, and updated simultaneously with weight. We call this method as OQFL(Optimized Quantization in Federated Learning). OQFL is a method of learning deep networks and quantization while maintaining security in a distributed network environment including edge computing. We introduce the OQFL method and simulate it in various Convolutional deep neural networks. We shows that OQFL is possible in most representative convolutional deep neural network. Surprisingly, OQFL(4bits) can preserve the accuracy of conventional federated learning(32bits) in test dataset.

Keywords—federated learning, OQFL, FPROPS, quantization

I. INTRODUCTION

Recently, the computing paradigm has shifted from centralization to distributed environment. In addition, a variety of mobile devices are emerging to meet user needs, and user-tailored services are becoming more important with the development of AI. Therefore, personal privacy is becoming very important. Federated learning is a distributed learning method that can preserve the user's data privacy. It became possible as the computing power of local devices increased. The local device is end device on the network. It can be internet of things, gateway, edge and user device. All data is stored in the local devices, not the central server, and the deep network is trained on the device. Therefore, unlike the general deep learning process, federated learning can preserve the user privacy completely because data is not leaked out of the user device during the learning process[1].

Deep network can be trained not only the data collected by the central server, but also the security critical data, so it can utilize all data. In deep learning, the amount and quality of training data is important. Federated learning, which can use data in local devices as well as central server data, has enormous potential in deep learning. Many studies have been conducted on resource allocation and communication efficiency, but studies related to deep network compression in federated learning process are weak.

Deep network compression is a necessary technology for 'On device AI', which is in the spotlight. It can make deep network faster inference with low capacity of user device. Particularly, among the compression field, quantization technology make the weight and activation of deep network in 2bits ~ 16 bits integer from 32bits float, thereby reducing FPROPS to make inference of deep networks faster. [2] quantized the weight of the existing 32 bits network into 2bits at the expense of accuracy loss. However, if only the weights are quantized, the FPROPS can't be significantly reduced. [3] significantly reduced FPROPS by quantization not only weights but also activations. By doing that, we can obtain quantized weights and activations, and by multiplying, faster inference is possible. Unlike the above method of quantization by analyzing the data distribution of weight and activation, [5] and [6] quantized the network by using back propagation algorithm as training new parameters. The new parameters updated by SGD([5],[6]). We thought this idea could be applied to federated learning.

In this paper, we propose a new learning method, OQFL which simultaneously performs network learning and quantization parameters learning using SGD(stochastic gradient descent) in a general federated learning process. Since all data is in user devices , the central server can't access the data. Therefore, user privacy can be completely preserved,

This work was partially supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-01287, Evolvable Deep Learning Model Generation Platform for Edge Computing) and supported by Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT)(2020-0-00364, Development of Neural Processing Unit and Application systems for enhancing AI based automobile communication technology)
*Dr CS Hong is the corresponding author.

and deep network and quantization parameters can be learned simultaneously by OQFL.

There are two main contributions in this paper.

1. We demonstrate a simple way to quantize deep network in general federated learning. The parameters for quantization are updated by SGD, and it performs simultaneously with deep network update in federated learning process.
2. We prove that OQFL can be applied various convolutional deep networks through experiment and, the 4bits deep network by OQFL can preserve accuracy of the 32bits deep network by conventional federated learning.

II. Related work

A. Federated Learning

There are several procedure in federated learning. Firstly, central server create a deep network and sends the deep network to user devices. Secondly, the deep network is trained in user device by the data in user device. Thirdly, the deep networks is sent to central server from all user devices. Lastly, all deep networks which trained in user devices is aggregated by central server, and the aggregated deep network is sent to user devices again. The aggregation formula is presented in the following equation[1].

$$w_{t+1} \leftarrow \sum_{n=1}^{N}(\frac{k_n}{k})w_{t+1}^n \quad (1)$$

In ①, k_n is the number of data of the nth user, k is the total number of data. w_{t+1}^n is the weight that trained by nth user device. w_{t+1} is the aggregated weight. [1] named above procedure as FedAvg. The reason the FedAvg algorithm is possible is that the deep network trained by back propagation algorithm by SGD [1]. Weight is updated to $w_{t+1} \leftarrow w_t - \gamma g_n$ by SGD and it can be expressed as $w_{t+1} \leftarrow w_t - \gamma \sum_{n=1}^{N}(\frac{k_n}{k})\, g_n$(FedSGD[1]) in distributed environment. g_n is the gradient of the deep network updated on the nth local device. In the deep network trained by SGD, it means that the FedSGD aggregating the mean of the gradients and the FedAvg aggregating the mean of the weights are the same in distributed learning.

Unlike regular learning, federated learning does not require centralized data collection. Since all data is not leaked within user devices, user privacy can be completely preserved.

B. Quantiztion

Deep network quantization is a way to reduce FLOPS for faster inference. [2] quantized the weight of the deep network into 2 bits at the loss of accuracy in the following way. They used two methods: deterministic and stochastic. Deterministic method is as follows:

$$w_b = \begin{cases} +1 & if\ w \geq 0 \\ -1 & otherwise \end{cases} \quad (2)$$

And stochastic is $w_b = +1\ with\ probability\ p = \sigma(w),\ -1\ with\ probability\ 1-p$. In addition, [3] reduced the FPROPS by quantization the weight of the deep network as well as activation into 2 bits, enabling faster than [2]. [4] quantized not only weight and activation but also gradient. Above papers don't use quantization parameters and trained deep networks by loss of quantized deep network.

Unlike the existing general method of minimizing the

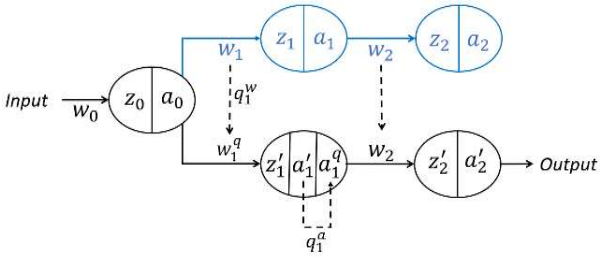

Fig. 1. Three layers neural network with quantization parameters

difference between the trained network and the quantized network, that is minimizing the quantization error, [5] used the loss in general network learning process by back propagation algorithm to learn quantization parameters. Quantization parameters consist of $\{c_{w_l}, d_{w_l}, c_{a_l}, d_{a_l}\}$. c means center of quantization interval, d means distance from the center(w is for weight, a is for activation and l is for the lth layer). Weights and activations are clipped or pruned if the value is outside the quantization interval range. Surprisingly their method in 4 or 5bits cases preserve the accuracy of the full-precision network. Also, the parameters are trained with weights of deep network by general task loss. [6] also, used learnable quantization parameters. Weights and activations are quantized by equation [6].

$$\bar{v} = \lceil clip(\frac{v}{q}, -Q_N, Q_P) \rfloor \quad (3)$$

In equation (3), v is weight or activation, and when quantize a deep network with $n$bits, $Q_N = 2^{t-1}, Q_P = 2^{t-1} - 1$ for weight and $Q_N = 0, Q_P = 2^t - 1$ for activation. $clip$ is $-Q_N$ if v/q is less than $-Q_N$, Q_P if it is greater than Q_P and v/q if it is between. $\lceil \rfloor$ is a rounding operation. q is the quantization parameter, and it is learnable by the gradient(straight through estimator[7], [8]) of equation (4)[6].

$$\partial\hat{v}/\partial q = \begin{cases} -v/q + \lceil v/q \rfloor & if\ -Q_N < v/q < Q_P \\ -Q_N & if\ v/q \leq -Q_N \\ Q_P & if\ v/q \geq Q_P \end{cases} \quad (4)$$

They also used gradient sale, and the performance of [6] is better than [5]. Of course, [6] and [4] quantized both weight and activation, and minimized the loss of accuracy of the quantized network using back propagation algorithm by SGD. We thought that the method of [6] could be applied to federated learning because quantization parameters can be learned by SGD.

III. Method

In this paper, the quantization method of [6] is used. In forward pass, v(weight or activation) is quantized to v_q by the quantization parameter q. The output is calculated from the quantized deep network and input data, and the loss is obtained. When forward pass, activation a^q and weight w^q are integers, so the quantization effect is maximized in the

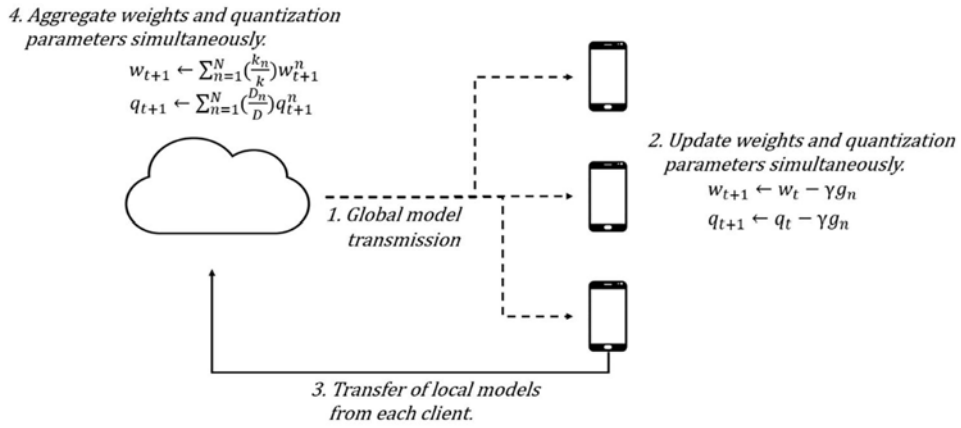

Fig. 2. OQFL process (the smart phone is the end of network, and it can be internet of things device, gateway, base station, UAV, edge and the cloud means central server and solid line is uplink and dotted line is downlink)

process of $a^q * w^q$. In backward pass, gradient is obtained by loss (SGD) and back propagation algorithm is executed. Not only the gradient of activations and weights, but also the gradient of quantization parameters is obtained. Fig 1 is a simplified representation of a quantized three layers neural network by q. The blue neurons are general 32bits neural network and the black neurons are quantized networks. After learning, only black neurons remain. The learnable parameters in Fig 1 are weights w_l and quantization parameters q_l. The first and last layers are not quantized. Quantization parameters are q_l^w for weights and q_l^a for activations. In Fig 1, q and w are updated as follows by chain rule.

$$q_1^{a+1} = q_1^a - \gamma \left(\frac{\partial L}{\partial a_2'} \frac{\partial a_2'}{\partial z_2'} \frac{\partial z_2'}{\partial w_2} \frac{\partial w_2}{\partial a_1^q} \frac{\partial a_1^q}{\partial q_1^q} \right)$$

$$q_1^{w+1} = q_1^w - \gamma \left(\frac{\partial L}{\partial a_2'} \frac{\partial a_2'}{\partial z_2'} \frac{\partial z_2'}{\partial a_1^q} \frac{\partial a_1^q}{\partial a_1'} \frac{\partial a_1'}{\partial z_1'} \frac{\partial z_1'}{\partial w_1^q} \frac{\partial w_1^q}{\partial q_1^w} \right) \quad (5)$$

$$w_1^{+1} = w_1 - \gamma \left(\frac{\partial L}{\partial a_2'} \frac{\partial a_2'}{\partial z_2'} \frac{\partial z_2'}{\partial a_1^q} \frac{\partial a_1^q}{\partial a_1'} \frac{\partial a_1'}{\partial z_1'} \frac{\partial z_1'}{\partial w_1^q} \frac{\partial w_1^q}{\partial w_1} \right)$$

The quantization parameters q can be updated $q_{t+1} \leftarrow q_t - \gamma\, g_n$ like the weight update $w_{t+1} \leftarrow w_t - \gamma\, g_n$, and the FedSGD algorithm of [1] can be applied in the form of $q_{t+1} \leftarrow q_t - \gamma \sum_{n=1}^{N}(\frac{k_n}{k})\, g_n$. Like the weight in [1], it can be expressed as follows.

$$q_{t+1} \leftarrow \sum_{n=1}^{N}(\frac{D_n}{D})q_{t+1}^n \quad (6)$$

In equation (6), D is the total number of data, D_n is the number of data the nth client has. q_{t+1}^n is the quantization parameter updated by n th client. q_{t+1} is the aggregated quantization parameter. The Since q for quantization of deep network is updated by back propagation algorithm by SGD, weights and quantization parameters are simultaneously updated by using loss in OQFL process. The entire process is represented in Fig 2. First, the central server creates a deep network and sends it to the learning participants. Second, participants train weight and quantization parameter of the deep network with minibatch using their own data. Third, The deep networks trained on the device are sent to the central server. Finally, weights and quantization parameters from devices are aggregated on the server.

Algorithm 1 shows aggregating weights and quantization parameters on the central server. N is the number of clients participating in learning, and P clients are randomly selected by the fraction of participation f for each epoch. Weight and quantization parameters of deep networks updated from clients aggregated simultaneously by $e_{t+1} = \sum_{n=1}^{N} \frac{D_n}{D} e_{t+1}^n$. e means all elements, including weight, bias, and quantization parameter.

Algorithm 1. Server : Weights and Quantization parameters Aggregation.

Server:

Initialize w_0, q_0, E : epochs

N : number of clients

f : fraction of participation

for e in range(E):

$P = \max(N * f, 1)$

$S \leftarrow$ random set of P clients

for s in S:

$w_{t+1}^s, q_{t+1}^s \leftarrow$Client($w_t, q_t, s$)

$e_{t+1} = \sum_{n=1}^{N} \frac{D_n}{D} e_{t+1}^n$

Algorithm 2 shows weights and quantization parameters update at client. We can use mini-batch by B. The client trains the deep network received from the central server with its data E times. L is the number of layers of the deep network, and $func$ is the activation function of relu, sigmoid, etc. The deep network quantized in forward pass, and we can get the loss by forward pass. The deep network, including quantization parameters is trained in { $learning\ network$ } part by the loss from forward pass. Of course, quantization parameters, weights and bias are trained at the same time. After algorithm 2, client send the information including weights, bias and quantization parameters to server. Then, the server execute algorithm 1 again.

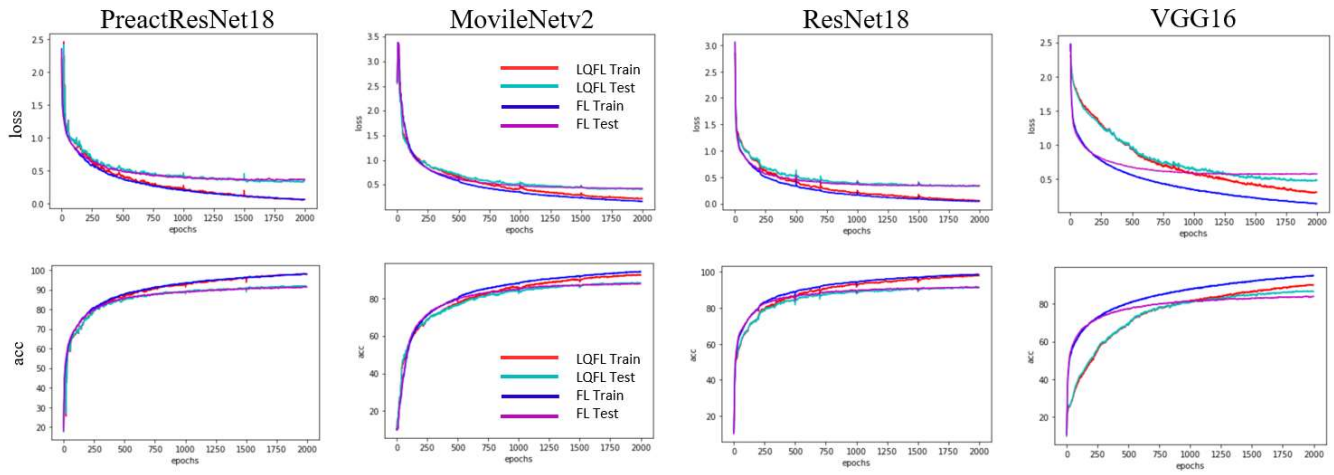

Figure 3. Accuracy and Loss in OQFL for convolutional deep neural networks

Algorithm 2. Client : Weights and Quantization parameters update.

Client(w, q, s):

Initialize B: batches

E: local epochs

L: number of layers

$func$: activation function(e.g. relu, sigmoid)

{ $forward\ pass$ }

$z_0 = forward(data_{input}, w_0)$

$a_0 = func(z_0)$

$a_0^q = a_0$

for l in range$(1, L-1)$:

$w_l^q = quant(w_l, q_l^w)$ ②

$z_l' = forward(a_{l-1}^q, w_l^q)$

$a_l' = func(z_l')$

$a_l^q = quant(a_l', q_l^a)$

$z_L' = forward(a_{L-1}^q, w_L)$

$a_L' = func(z_L')$

$loss \leftarrow CorssEntropy(y, a_L')$

{ $learning\ network$ }

for e in range(E):

for b in B:

$$w = w - \gamma(\frac{\partial loss}{\partial w}; b)$$

$$q^a = q^a - \gamma(\frac{\partial loss}{\partial q^a}; b)$$

$$q^w = q^w - \gamma(\frac{\partial loss}{\partial q^w}; b)$$

return w, q

IV. RESULT

In this paper, we used cifar10 dataset and experimented in iid environment. It is assumed that all data is stored on the user devices, not central server. We simulated on pytorch and used cross entropy function as loss, and momentum is 0.9. The total number of clients are 100, f(fraction of participation) is 0.1, E(local epochs) is 5, batch size is 100, local learning rate is 0.01(learning rate of VGG16 is 0.001).

We used ResNet[9], PreactResNet[10], MovileNetv2[11], VGG[12] for deep convolutional neural networks. We quantized all networks to 4bits from 32bits full precision in OQFL process. In Fig 3. All convolutional deep neural networks are trained with similar slopes in OQFL and conventional federated learning(FL). You can see that OQFL works well and it can be applied well to various convolutional deep neural networks. However, if you look closely at the graph, you can see the serrated shape in the graph of OQFL, unlike FL. OQFL is more unstable than FL due to quantization, but instability doesn't affect training results. We can get 4bits network after OQFL. The 4bits network has much faster inference comparing to 32bits full-precision network. Therefore, the quantized networks by OQFL will run more smoothly on local devices with low computing power. In addition, we can preserve user privacy while learning network.

Surprisingly, the 4bits networks by OQFL preserves the accuracy of the 32bits full-precision network by FL. FL has slightly better performance than OQFL in train dataset. However, OQFL has better performance in most networks in test dataset. In Table 1, all four networks(ResNet[9], PreActResNet[10], MovileNetv2[11], VGG[12]) have same result that the accuracy of OQFL can preserve the accuracy of conventional federated learning. We found that the difference between the performance in test dataset and the performance in train dataset is lower in OQFL than FL. This means that overfitting in FL occurs more severely than OQFL.

We found that OQFL doesn't work when the total number of users are 50 and f is 0.1 in ResNet18[9] case. We guessed the reason. There are 50000 training images in cifar10. In the same environment above experiment, the first case is that each user has 100 images, when the total number of users that participate in OQFL is 50. the second case is that each user has 200 images when the total number of users is 100. The deep network is trained all of the data on user device 5times. The deep network is trained more images in the first case.

Table 1.Conventional Federated Learning vs OQFL for various convolutional deep neural networks

		Test Dataset		Train Dataset	
Network	Method	Best Loss	Best Accuracy	Best Loss	Best Accuracy
ResNet18	FL(32bits)	0.328594	91.73	0.039707	98.742
	OQFL(4bits)	0.321792	91.6	0.054859	98.126
PreActResNet18	FL(32bits)	0.347764	91.51	0.049984	98.292
	OQFL(4bits)	0.319505	92.04	0.05699	98.064
MovileNetv2	FL(32bits)	0.425524	87.86	0.165942	94.306
	OQFL(4bits)	0.407621	88.49	0.215707	92.652
VGG16	FL(32bits)	0.568557	84.11	0.138565	95.184
	OQFL(4bits)	0.469108	86.96	0.300512	90.216

Therefore, in second case, the difference between the received deep network from central server and the deep network after training in user device is more than first case. In other words, all deep networks trained on all user devices are very different in second case then, error occur during the aggregation by central server. We found that the quantization parameter is very sensitive in aggregation of OQFL. We will experiment with different learning rates for weight and quantization parameter, and find the optimal learning rates.

V. Conclusion and Future Work

We proposed a new method to quantize deep network in federated learning process. There is no way to quantize using data in general federated learning. However, the proposed method can quantize it from 32bits to 2, 3, 4bits. The quantization parameters are easily applicable to federated learning, because they are trained by SGD. We can get a quantized deep network with much faster inference after federated learning. We can simultaneously update deep network and quantization parameters, and we can get a quantized deep network with much faster inference after OQFL. Surprisingly, the deep network after OQFL preserve the accuracy of the deep network after federated learning. In particular, OQFL has better performance than conventional federated learning in the most networks in test dataset.

In next study, We are going to experiment OQFL in non-iid environment and compare with iid. In addition, we will quantize it to 2, 3bits and measure how much it preserves the accuracy of full-precision networks. OQFL has something to supplement. There are additional parameters q for quantization deep network, that means in OQFL, the edge needs more computing power comparing to conventional federated learning. Also, as more information is transmitted, it can cause a communication bottleneck. To solve this problem, we are going to quantize the information from devices to central server in OQFL process for communication efficiency.

VI. References

[1] McMahan, H. B., Moore, E., Ramage, D., & Hampson, S. (2016). Communication-efficient learning of deep networks from decentralized data. arXiv preprint arXiv:1602.05629.

[2] Courbariaux, M., Bengio, Y., & David, J. P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. In Advances in neural information processing systems (pp. 3123-3131).

[3] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). Binarized neural networks. In Advances in neural information processing systems (pp. 4107-4115).

[4] Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., & Zou, Y. (2016). Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160.

[5] Jung, S., Son, C., Lee, S., Son, J., Han, J. J., Kwak, Y., ... & Choi, C. (2019). Learning to quantize deep networks by optimizing quantization intervals with task loss. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4350-4359).

[6] Esser, S. K., McKinstry, J. L., Bablani, D., Appuswamy, R., & Modha, D. S. (2019). Learned step size quantization. arXiv preprint arXiv:1902.08153.

[7] Bengio, Y., Léonard, N., & Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432.

[8] Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., & Zou, Y. (2016). Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160.

[9] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[10] He, K., Zhang, X., Ren, S., & Sun, J. (2016, October). Identity mappings in deep residual networks. In European conference on computer vision (pp. 630-645). Springer, Cham.

[11] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).

[12] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.