Network Update Compression for Federated Learning

1st Birendra Kathariya
University of Missouri-Kansas City
Kansas City, USA
bkkvh8@umsystem.edu

2nd Li Li
University of Missouri-Kansas City
Kansas City, USA
lizhu@umsystem.edu

3rd Zhu Li
University of Missouri-Kansas City
Kansas City, USA
lil1.umsystem.edu

4th Lingyu Duan
Peking University
Beijing, China
lingyu@pku.edu.cn

5th Shan Liu
Tencent America
Palo Alto, USA
shanl@tencent.com

***Abstract*—In federated learning setting, models are trained in a variety of edge-devices with locally generated data and each round only updates in the current model rather than the model itself are sent to the server where they are aggregated to compose an improved model. These edge devices, however, reside in highly uneven nature of network with higher latency and lower-throughput connections and are intermittently available for training. In addition, a network connection has an asymmetric nature of downlink and uplink. All these contribute to a major challenge while synchronizing these updates to the server.**

In this work, we proposed an efficient coding solution to significantly reduce uplink communication cost by reducing the total number of parameters required for updates. This was achieved by applying Gaussian Mixture Model (GMM) to localize Karhunen–Loève Transform (KLT) on inter-model subspace and representing it with two low-rank matrices. Experiments on convolutional neural network (CNN) models showed the proposed model can significantly reduce the uplink communication cost in federated learning while preserving reasonable accuracy.

***Index Terms*—federated learning, network-update compression, Karhunen–Loève Transform (KLT)**

I. Introduction

Edge devices such as cellphones, surveillance cameras, sensors equipped in vehicles etc. have access to unfathomable amount of data. Learning a better model from these valuable data is thus suitable to improve upon user experience and help power more intelligent applications. However, gathering these data to a centralized location for training is not practical mainly due to two reasons: data are privacy sensitive and communication cost involved while uploading voluminous data in an unreliable bandwidth limited network.

However, with Federated Learning [1] [2], we avoid these limitations by allowing models to train locally on edge devices (clients) and collect only model-updates, differential value of current updated model in clients and global shared model distributed to client at previous round, to the server thereby decoupling the edge devices with the server and all the other devices ensuring the privacy of such sensitive data. Simultaneously, it also reduces the uplink communication as model-update which are smaller in size are communicated rather than large chunk of training data. Visual representation of Federated Learning is shown in Fig. 1.

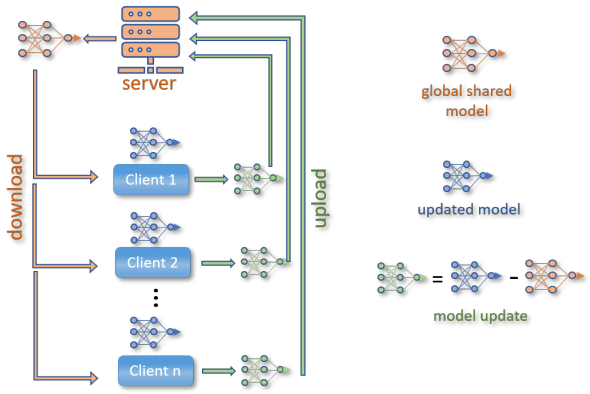

Fig. 1. Visual Representation of Federated Learning

Federated Learning, however when considered across millions of edge devices, poses multiple practical challenges. The primary constraint is the uneven nature of network connections these devices reside. Uplink communication is relatively slower and any device residing on slower and unreliable network will experience higher latency and are intermittently available for training. This will suffer overall learning process as communication rounds between server and clients decreases. One way to improve the learning process is by efficiently compressing the model-updates thus easing up an uplink communication thereby increasing the communication rounds.

In this work, we present an efficient method to reduce uplink communication involving CNN-model for federated learning. We learn multiple GMM based localized KLT models in convolutional layer update. Low-rank approximation which preserves sufficient information of these KLT models are then communicated to the server thereby reducing total numbers of update parameters related to the convolutional layers.

978-1-7281-8068-7/20/$31.00 ©2020 IEEE

II. Related Work

There have been plethora of work [3] [4] [5] on compression of Deep Neural Network (DNN) model. However, federated learning emerged very recently and very few work can be found on model-update compression. These works, mostly burrowed concepts from DNN model compression and can be broadly categorized as below.

Sparsification : Work that adopts gradient sparsification approach truncates smaller gradients by hard-thresholding and transmits only the larger important ones. [6] used this approach and reduced the gradient exchange to almost 99% at 0.3% BLEU score loss while 22% speed gain. Similarly, [7] also used same approach but rather than throwing away smaller gradient they were accumulated until they become large enough to send eventually sending all gradients overtime. Same approach was also adopted by [8] while focusing on sparsity recovery.

Quantization : This category of work follows gradient quantization to low-precision values. *TernGrad* [9] quantized the gradient to three numerical values {-1, 0, 1}. Similarly, *DoReFa-Net* applied 1, 2 and 6 bit quantization to weights, activations and gradients respectively. *1BitSGD* [10] took an aggressive approach to quantize gradient just to 1-bit and effectively reduce communication of DNN for acustic modelling. *QSGD* [11] proposed *Quantized SGD (stocastic gradient descent)* which allows user to trade off between accuracy and gradient precision. [12] utilized same quantization approach and conducted an extensive experimentation to reduce inter model redundancies for various application scenarios.

Low-rank Approximation : In this approach the number of parameters to be communicated is reduced by low-rank approximation of gradient. Server reconstructs to full rank gradient from the limited information sent by the client. [13] used similar approach where each layer gradients is decomposed into two low-rank matrices. However, only one low-rank matrix is optimized and send to the server and other is either fixed or compressed in the form of random seed.

III. Proposed Method

Federated Learning proceeds by distributing a global shared model from server to n participating edge-devices at time $t = 0$. These models in their host devices are trained in parallel with locally generated data. After a fixed interval t_c, model-updates from a subset of available participating devices $S_t \leq n$ are collected to a server for aggregation [14].

We considered a convolutional-layer weights $W \in \mathbb{R}^{f\times c\times k_1\times k_2}$, where $k_1 \times k_2$ is kernel size, c is number of channels and f is number of filter bank, has a 2D representation $W \in \mathbb{R}^{d_1\times d_2}$, where $d_1 = c \times f$ and $d_2 = k_1 \times k_2$. Let at round t, S_t clients have updated-models W_t^i, $i \in S_t$, which are results of multiple steps of stochastic gradient descent (SGD) on client's local dataset. The updates of S_t clients, now can be written as $H_t^i = W_t^i - W_{t-1}$, $H_t^i \in \mathbb{R}^{d_1\times d_2}$, where W_{t-1} is a convolutional-layer weight of global shared model at previous round $t-1$.

A. Learning GMM based localized KLT Model

Deeper layer of CNN-models exhibits large number of filters. Learning single KLT model on a whole block of H_t^i could incur large accuracy loss. To circumvent this issue, we learned multiple KLT models on H_t^i. However, we employed GMM to optimally localize these models in a distribution of d_1 weight-updates. GMM uses famous *Expectation-Maximization Algorithm* [15] to learn the model parameters that best explains multivariate data.

Suppose, GMM learn M gaussian-models that best fit updates H_t^i. These models can be represented by three parameters $G_c : \{\mu_c, Cov_c, L\}$, $c \in M$, where μ_c is centroid of a distribution, Cov_c is its covariance and L is likelihood-probability. $L \in \mathbb{R}^{d_1\times M}$ represents how likely samples of H_t^i are related M gaussian-models. Since we wanted to represent data points in H_t^i by M models, we used index to the highest likelihood in a row of L to assign corresponding data-point in H_t^i a label $I = \{x \in \mathbb{R}^{d_1} : 0 \leq x \leq M\}$ for a model.

B. KLT Low Rank Approximation

Let $h_{t,c}^i \subset H_t^i$, $h_{t,c}^i \in \mathbb{R}^{d\times d_2}$, $d < d_1$ is a group of data point represented by model G_c. For convenience, we denote $h_{t,c}^i$ as h_c. The Cov_c parameter of G_c is the covariance of h_c. We perform singular-value decomposition (SVD) on covariance matrices Cov_c using (1) to get corresponding KLT models.

$$Cov_c = \Phi_c \Lambda \Phi_c^T \tag{1}$$

Here, $\Phi \in \mathbb{R}^{d_2\times d_2}$ is an eigen-vector and $\Lambda \in \mathbb{R}^{d_2}$ is eigen-value. Next, we project h_c to a rank k matrix using (2).

$$P_c = h_c \times \phi_c \tag{2}$$

where, $\phi_c \in \mathbb{R}^{d_2\times k}$ is Φ with 1^{st} k columns and $P_c \in \mathbb{R}^{d_1\times k}$ is projected matrix.

C. Full Rank KLT Reconstruction

Server should receive coefficient $\phi_c \in \mathbb{R}^{d_2\times k}$, projection matrix P_c and mean μ_c for all KLT models $c \in M$ and for all convolutional layers. A full-rank reconstruction for data-points h_c represented by a k rank KLT-model is achieved using (3).

$$\hat{h_c} = P_c \times \phi_c^T + \mu_c \tag{3}$$

where, $\hat{h_c} \in \mathbb{R}^{d_1\times d_2}$ is full-rank approximation of h_c. This is repeated for all KLT-models and for all layers. Once, updates in a convolutional layer are approximated, label I is used to reorder them back to their original index position along 1^{st} dimension. Let $\hat{H_t^i} \in \mathbb{R}^{d_1\times d_2}$ be the reconstructed model-update for a convolutional layer in the server. Next step is to aggregate these model-updates received from S_t clients layer-wise.

D. Update Aggregation

Server aggregates all the reconstructed model-updates $\hat{H_t^i}$ received from S_t clients and generates new global shared-model W_t. This aggregation and generation of new model is carried out as in (4) [13].

$$W_t = W_{t-1} + \eta_t H_t, \qquad H_t := \frac{1}{S_t} \sum_{i \in S_t} \hat{H}_t^i \tag{4}$$

Here, H_t is aggregated update, W_t is a new shared-model and η_t is a learning rate chosen by server. Finally, updated convolutional layer weights W_t are reshaped back to their original dimensions of $f \times c \times k_1 \times k_2$.

E. Parameter Scaling and LZMA Coding

For update reconstruction in server, four parameters: coefficient ϕ_c, projected matrix P_c, mean μ_c and gaussian model label I, are needed to be communicated. We utilized Lempel–Ziv–Markov chain algorithm (LZMA) [16], a dictionary based lossless compression scheme, to further compress these parameters before sending them to the server. However, coefficient, projection matrix and mean are fractional values in range $\{-1, 1\}$. We first converted these values to integer values using (5).

$$A_s = round(a \times 10^s) \tag{5}$$

where, a is a fractional value and A_s is an integer after scaling. By changing s, we allowed integer value to scale with digits truncated at s decimal place.

In the server, decoded scaled integer A_s is converted back to fractional value using (6).

$$\hat{a} = A_s \times 10^{-s} \tag{6}$$

F. Model-update compression for Dense and BatchNorm Layer

All update parameters related to fully connected and batch-normalization layers as well as biases from all layers were subjected to scaling using (5) and LZMA coding as described in previous subsection.

IV. Experimental Setup and Results

We considered two popular CNN models: vgg16 and resnet18 with and without batch-normalization (BatchNorm) respectively for evaluation of our work. Both models were trained on cifar10 dataset. We setup federated learning with five edge-devices thus cifar10 dataset was splitted into five non-overlapping subsets as local dataset. We trained vgg16 models for 150 epochs and resnet18 models for 250 epochs with local dataset allocated for respective edge-devices. Updates of these independent models were pushed to the server every 10 epochs for aggregation. For simplicity, we set learning rate η_t in (4) to 1 and utilized synchronous aggregation scheme by considering all available devices for aggregation. After aggregation, new shared model was validated with the test samples from cifar10 dataset. All the trainings were conducted with batch-size of 128, learning-rate of 0.05, momentum of 0.9 and weight-decay of 0.0005.

Baseline of this work transmitted the model updates with 32-bits. Vgg16 has 13 convolutional layers (3x3 kernels) and 3 dense layers with total of 15245130 (without BatchNorm) parameters. Similarly, resnet18 has 17 convolutional layers (3x3 kernels) and 1 dense layer with total of 11183562 (with BatchNorm) parameters. The average size of updates for vgg16 model transmitted from five edge-devices per round was 15245130×32 bits (465.24 Mbits) and for resnet18, it was 11183562×32 bits (341.29 Mbits). Resnet18 is smaller than vgg16 even though it has 18 layers (plus BatchNorm) whereas vgg16 has only 16 layers (and no BatchNorm). Most of the parameters are concentrated in dense layers of vgg16.

We first fixed the number of KLT models M for all convolutional layers proportional to the number of kernel d_1 and is presented in Table I. Then multiple experiments were conducted where we chose different scale value s from 4 to 2 while adjusting rank $k < d_2 = 9$ of KLT models. The chosen s and k value with corresponding converged mean average precision (mAP) and average of compressed update size (Mega Bits) from all five edge-devices per round of aggregation are presented in Table II. The reported mAP values are from the global shared model after completion of training. We also reported mAP of the global shared model for all rounds of aggregation which are plotted in Fig. 2 and 3 for vgg16 and resnet18 respectively. Reduction in model-update size compared to the baseline is reported as percentage in the last column of Table II.

TABLE I
Number of Kernels (d_1) and chosen number of KLT models (M)

d_1	4096	8192	16384	32768	65536	131072	262144
M	1	2	4	4	4	8	16

Rank k of KLT model is presented as lists (13 values for vgg16 and 17 values for resnet18) in the 4^{th} column of Table II, each value from left to right is chosen for a convolutional layer from depth 1^{st} to last. We chose smaller k for deeper layer. Moreover, the first layer was not subjected to KLT modelling but only scaling and LZMA coding similar to biases and dense layer. This is because the weights in the deeper layer contribute lesser on overall accuracy and vice-versa. Thus updates from deeper layers were represented with KLT models with smaller rank while from shallower layer with larger rank. Also, distortion in bias has greater impact on accuracy compared to weight. Thus, we chose larger scale value $s = 6$ for bias.

We made 4 different selections of s and k for vgg16 and 3 different selections for resnet18 which are reported in Table II. Smaller s value encoded model-update parameters and KLT model parameters with lesser precision and provided larger compression but also resulted in larger loss in accuracy. We minimized this loss by increasing k for all layers, however, without much burden in the communication bandwidth. In the last column of Table II we noticed an increase in compression when s is decreased from 4 to 2 without any significant loss in test accuracy (mAP). Further decreasing s to 1 failed models to learn thus we excluded results for $s = 1$.

TABLE II

TEST ACCURACY (MAP) AND COMPRESSION (MBITS) PERFORMANCE FOR DIFFERENT SELECTION OF s AND k. REDUCTION (%) IS PERCENTAGE DECREASE OF MODEL-UPDATE WITH RESPECT TO THE BASELINE.

Model	S.No.	Scale (s)	KLT Model Rank (k)	mAP	Size (MBits)	Reduction (%)
vgg16	baseline	-	-	87.76	465.24	-
	test 1	4	*, 5, 5, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3	87.68	36.51	92.15
	test 2	4	*, 6, 6, 5, 4, 3, 3, 3, 3, 3, 2, 2, 2	87.97	32.53	93.00
	test 3	2	*, 6, 6, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4	87.53	6.23	98.66
	test 4	2	*, 7, 7, 6, 6, 5, 5, 5, 5, 5, 5, 5, 5	87.72	6.44	98.61
resnet18	baseline	-	-	92.48	341.29	-
	test 1	4	*, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3	91.55	19.55	94.14
	test 2	3	*, 7, 7, 6, 6, 6, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4	91.81	10.38	96.95
	test 3	2	*, 6, 6, 6, 6, 6, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4	91.02	2.43	99.28

$*$ KLT modelling not applied in the 1^{st} cnn-layer.

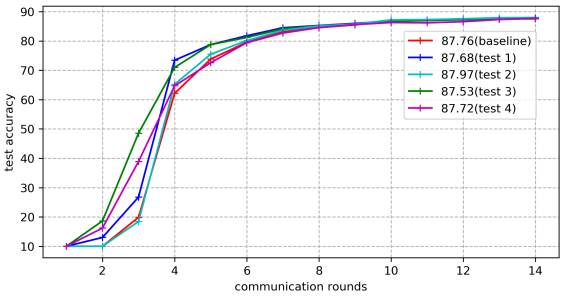

Fig. 2. Test Accuracy vs. Communication rounds for vgg16 trained with cifar10 dataset on five-edge devices

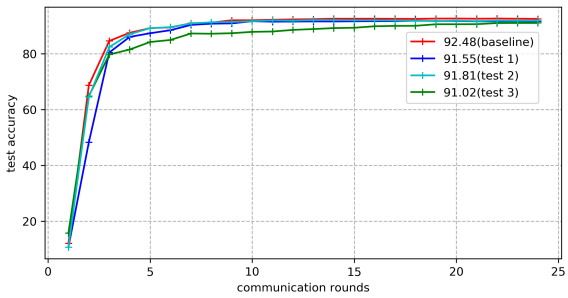

Fig. 3. Test Accuracy vs. Communication rounds for resnet18 trained with cifar10 dataset on five-edge devices

V. CONCLUSION

In this work we explored model-update compression in federated learning setting and presented a method that can efficiently reduce uplink communication by reducing the total numbers of parameters to be communicated. This was carried out by an inter-network and intra-update prediction scheme. Inter-network prediction was achieved through network weight differentiation. Similarly, intra-update prediction was achieved by learning multiple localized KLT models using gaussian-mixture-model (GMM) and compacting the energy in each kernel-update by projecting KLT models into a reduced subspace. Eventually, KLT model parameters and update-parameters from bias and dense layer were further compressed using LZMA coding. We needed to make a careful selection of three parameters: numbers of KLT models per convolutional layer, rank of the KLT model per convolutional layer and scale value to achieve best accuracy and compression performance.

REFERENCES

[1] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *CoRR*, vol. abs/1602.05629, 2016.

[2] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *CoRR*, vol. abs/1902.04885, 2019.

[3] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," *CoRR*, vol. abs/1512.06473, 2015.

[4] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems 5* (S. J. Hanson, J. D. Cowan, and C. L. Giles, eds.), pp. 164–171, Morgan-Kaufmann, 1993.

[5] R. Rigamonti, A. Sironi, V. Lepetit, and P. Fua, "Learning separable filters," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2754–2761, June 2013.

[6] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *CoRR*, vol. abs/1704.05021, 2017.

[7] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *CoRR*, vol. abs/1712.01887, 2017.

[8] R. Garg and R. Khandekar, "Gradient descent with sparsification: an iterative algorithm for sparse recovery with restricted isometry property," in *ICML*, 2009.

[9] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," *CoRR*, vol. abs/1705.07878, 2017.

[10] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns," in *INTERSPEECH*, 2014.

[11] D. Alistarh, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: randomized quantization for communication-optimal stochastic gradient descent," *CoRR*, vol. abs/1610.02132, 2016.

[12] Z. Chen, L. Duan, S. Wang, Y. Lou, T. Huang, D. O. Wu, and W. Gao, "Toward knowledge as a service over networks: A deep learning model communication paradigm," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1349–1363, 2019.

[13] J. Konecný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *CoRR*, vol. abs/1610.05492, 2016.

[14] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* (A. Singh and J. Zhu, eds.), vol. 54 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 1273–1282, PMLR, 20–22 Apr 2017.

[15] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, vol. 39, no. 1, pp. 1–38, 1977.

[16] "Lzma coding." https://www.7-zip.org/sdk.html. Accessed: 2020-7-6.