Pace Control via Adaptive Dropout for Federated Training

A Work-in-Progress Report

Feiyang Wang[†], Xiaowei Shang[◇], Jianchen Shan[*], Xiaoning Ding[◇]
[†]*Department of Mathematics and Computer Science, Rutgers University - Newark Campus*, Newark, NJ, USA
[◇]*Department of Computer Science, New Jersey Institute of Technology*, Newark, NJ, USA
[*]*Department of Computer Science, Hofstra University*, Hempstead, NY, USA

***Abstract*—This paper proposes a neuron drop-out mechanism to control the training paces of mobile devices in federated deep learning. The aim is to accelerate the speed of local training on slow mobile devices with minimal impact on training quality, such that slow mobile devices can catch up with fast devices in each training round to increase the overall training speed. The basic idea is to avoid the computation of some neurons with low activation values (i.e., neuron dropout), and dynamically adjust dropout rates based on the training progress on each mobile device. The paper introduces two techniques for selecting neurons, LSH and Max Heap, and a method for dynamically adjusting dropout rates. It also discusses a few other approaches that can be used to control training paces.**

I. Introduction

Federated Learning has attracted considerable attention. It utilizes a large-scale distributed system crossing cloud and millions of user mobile devices to train a model (e.g., a deep neural network) with the data on these devices. Compared to other distributed training systems, user data does not leave their devices with federated learning, which eliminates users' privacy concerns. For the federated learning of neural network, after the initial network structure is established, a central server running in the cloud is used to start training rounds and coordinate the training in mobile devices. Specifically, in each round, the server firstly selects a certain number of available participating mobile devices, configures the schedule, and broadcasts neural network parameters to these mobile devices to initiate their local training. Once local training finishes, the server aggregates the local model gradients on the mobile devices to update the global neural network parameters, which are carried to the next round.

One of the key issues in federated learning is to deal with the low and heterogeneous computing capabilities on mobile devices. Compared to conventional distributed systems for deep learning, which use a cluster of high-end servers to train a neural network, federated learning relies on the computing resources on mobile devices. However, the computing resources available to federated training (e.g., available CPU/GPU cycles, free memory, remaining battery life, etc) are usually limited on mobile devices, and vary across different devices and over time. This significantly limits training speed.

This paper explores the techniques that can accelerate federated training by controlling the training paces on mobile devices. Without a mechanism controlling the paces, in each training round, fast devices may spend long time waiting for slow devices to finish their local training in the current round before moving to the next round; thus, the training speed is largely throttled by slow devices. Pace control is to substantially increase the training speed on slow devices with minimal impacts on training quality. It is partly motivated by heterogeneous distributed learning [1], which is designed to coordinate the learning on heterogeneous resource-constrained devices. With heterogeneous distributed learning, each computing device employs a neural network with a size that fits its computing capability; i.e., resource-constrained devices can implement less accurate but less complex neural networks than other devices, allowing them to cache up with other devices. However, heterogeneous distributed learning cannot be utilized in federated learning, because it requires devices to share sample parameters, which incur privacy risks and high communication overhead.

This paper proposes a neuron drop-out mechanism for pace control in federated deep learning, which can maximize the total utilization of participating devices to increase training speed. This mechanism does not require participating devices to share their data. The basic idea is to avoid the computation of a proportion of neurons on slow devices, and dynamically adjust the proportion based on the training progress on each mobile device. The paper discusses how to select neurons to drop out and how to adjust dropout rates.

II. Two Neuron Dropout Techniques

In deep learning, dropping out some neurons in a neural network can substantially reduce computation workload and mitigate over-fitting. Previous works have shown that keeping only the neurons with high activation values (called active neurons) in training can achieve higher accuracy than normal dropout [2], [3]. The paper combines two methods to select active neurons. One is Locality Sensitive Hashing (LSH) [4]–[6], which is highly efficient and fits best slow mobile devices because of its high selectivity. The other is a max heap dropout method, which can select active neurons at a relatively low selectivity for the mobile devices with moderate speeds. This section introduces these techniques.

A. LSH Dropout in Sub-Linear Time

When training a neural network, forward propagation can be described as:

$$a^l = f(W^l a^{l-1} + b^l) \tag{1}$$

978-1-7281-8266-7/20/$31.00 ©2020 IEEE
DOI 10.1109/IEEECloudSummit48914.2020.00036

where a^l and a^{l-1} are the output vectors for layers l and $l-1$ respectively, b^l is the bias vector, W^l is the weight matrix, and $f(\cdot)$ is the activation function (e.g., sigmoid, tanh, or ReLU). Because $f(\cdot)$ is monotonically increasing when its input is in $(0, 1)$, a^l is proportional to wa^{l-1}. Thus, finding the neurons with high activation values can be considered as a maximum inner product search problem (MIPS), i.e., searching for the neurons (n_i) with the highest inner products between a^{l-1} and their weights w_i^l, i.e, $a^{l-1}w_i^{l^T}$.

Note that the euclidean distance between w_i^l and a^{l-1} is $\left\|a^{l-1} - w_i^l\right\|^2 = \left\|a^{l-1}\right\|^2 + \left\|w_i^l\right\|^2 - 2a^{l-1}w_i^{l^T}$ for a neuron n_i. After the inputs are normalized with weight normalization [7] for all layers, $\left\|a^{l-1}\right\|$ and $\left\|w_i^l\right\|$ are both constants. Thus, the euclidean distance is $\left\|a^{l-1} - w_i^l\right\|^2 = C - 2a^{l-1}w_i^{l^T}$, where C is a constant. This converts the MIPS problem into to a nearest neighbor search problem, i.e., to find the neurons with parts of their weight vectors similar to their inputs a^{l-1}.

The nearest neighbor search problem can be efficiently solved utilizing Locality Sensitive Hashing (LSH) family function $h(\cdot)$. The function calculates the colliding probability of two vectors a and w, and guarantees that the possibility monotonically increases with the similarity between the vectors. Thus, it can be used to query the approximately highly similar vectors.

For a layer l, L hash tables are constructed to find the active neurons set AS_l in sub-linear time. These tables are queried with a^{l-1} being the querying vector. For each table, a LSH hash function $h(\cdot)$ is formed by concatenating K independent hash bits $\{h_1(\cdot), h_2(\cdot), ..., h_k(\cdot)\}$. All the neurons with their weight vectors corresponding to the hash key $h(a^{l-1})$ in these tables are selected as active neurons.

For LSH, K and L are key parameters. As each hash bit in the hash key can be viewed as the existence of a certain "feature" of the querying vector, with a higher K, the items in the corresponding hash buckets are similar to the querying vector with a higher probability. Thus, increasing K lowers "false-positive" in finding high activation neurons. Increasing L helps reducing "false-negative", because collecting all the neurons found in more hash tables reduces the chance that a neuron with high activation value is missing.

B. Max-Heap Dropout in Linear Time

As an alternative way to select active neurons, after the activation values have been computed, we can build a max heap with these values, with which the highest activation values (and the corresponding neurons) can be quickly located. The max heap method provides the flexibility of selecting high activation values at any given selectivity. To select K highest value neurons from N neurons (selectivity is K/N), the time complexity is $O(N + k * LogN)$. When $k < c * N/log(N)$, where c is constant, the time complexity becomes $O(N)$.

III. Pace Control with Adaptive Neuron Dropout

Figure 1 shows how the training paces are controlled. During the training, a dropout rate is determined for each mobile device based on its progress (Subsection III-A). Then,

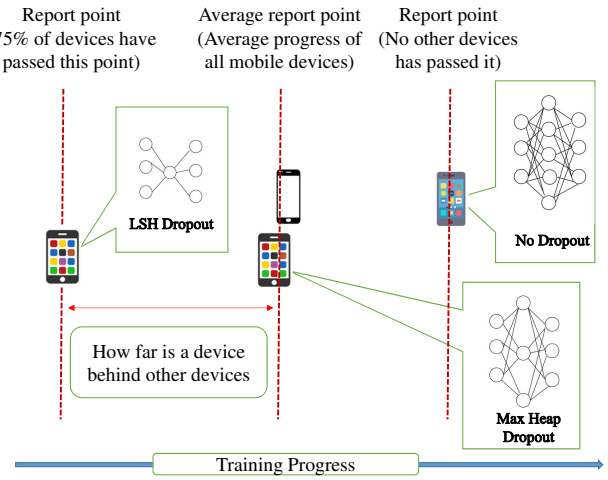

Fig. 1. Page Control with Dynamic Neuron Dropout

a technique, either LSH or Max Heap, is selected to dropout neurons (Subsection III-B), LSH for high dropout rates and Max Heap for low dropout rates.

A. Dynamically Adjusting Dropout Rates

The dropout rate of a mobile device is determined by two factors: 1) the relative position of a mobile device, i.e., whether it is a straggler or it is proceeding faster than most of others; and 2) the training speed difference between the mobile device and others.

As shown in Figure 1, to determine the first factor, in each round, we set a fixed number of report points as the landmarks for progress. When a mobile device reaches a report point, it reports this progress to the server, and the server sends back how many participating mobile devices (in percentage) have passed this report point. The larger the percentage is, the larger the dropout rate should be.

To determine the second factor for a mobile device, we first calculate an average value (named *average report point*) of the latest report points that have been passed by all mobile devices. Then, we compute the difference between the average report point and the current point as the indicator.

Note that, for the mobile devices that are much faster than other mobile devices, their dropout rates are very low. There is no need to dropout any neurons for these mobile devices, because the overall training progress is not determined by these mobile devices.

B. Neuron Selection with LSH and Max-Heap

Though LSH drop out is highly efficient, two issues must be addressed to be used to dynamically adjust the training paces on mobile devices. One is how to maintain high accuracy, which is a key requirement of pace control. The other is how to further improve efficiency to minimize the resource consumption on mobile devices. We address these issues by combining LSH dropout, Max-Heap dropout and exponential decay, as we introduce below.

With LSH, after hash tables have been built with pre-selected K and L parameters, the selectivity is roughly determined. Because pace control needs to dynamically change the dropout rate, it is possible that excessive neurons or insufficient neurons are selected by LSH, or in other word, the number of neurons selected by LSH can not match the number of neurons determined by the desired dropout rate. This issue can be solved as follows. For a given dropout rate, when enough active neurons are chosen by LSH, we terminate neurons query with LSH; when LSH cannot choose enough active neurons, neurons taken from a random array created beforehand can be used as supplement.

However, the above solution can reduce accuracy. Especially, when the number of neurons selected by LSH is significantly lower than the number of neurons determined by the desired dropout rate, too many random neurons would be chosen. In these cases, to guarantee accuracy, the top-k max heap dropout algorithm is more preferable, though the $O(N)$ time complexity with Max-Heap is higher than that of LSH dropout.

With LSH, the cost of updating the hash tables may be high. After updating gradients, weights change, so do their hash codes. Thus, hash tables must be updated to reflect these changes. This incurs high cost, especially on the mobile devices that are short of resources or filled with data. To address this issue, we borrow the exponential decay idea in [8], which exponentially increases the number of iterations between consecutive hash table updates. However, even with exponential decay, in the initial stage, hash tables must be updated frequently, and the costs may still exceed computing all the neurons in the network. Thus, LSH dropout is used only when the convergence of the model starts to slow down after the initial stage and when hash tables can be updated infrequently. Max Heap dropout can be used when LSH is inactive.

IV. Alternative Approaches for Pace Control

The key to effective pace control is to adjust the computation workloads on slow mobile devices with minimal impacts to training quality. While this goal can be achieved by dropping out neurons with low activation values, it can also be achieved in other ways, for example reducing the number of data required in training, and offloading computation to other devices. This section discusses and compares two possible approaches in these two directions: common data dropout, and split learning. Common data dropout looks to be more promising. The combination of this approach with neuron dropout can be our future work.

A. Common Data Dropout via LSH

Mobile users generate a huge amount of data. Not all the data can effectively contribute to model training. For example, users may have identical or similar data, because they may share photos/video clips or download similar contents from internet. In federated learning, local training is performed independently on each individual mobile device. A data sample

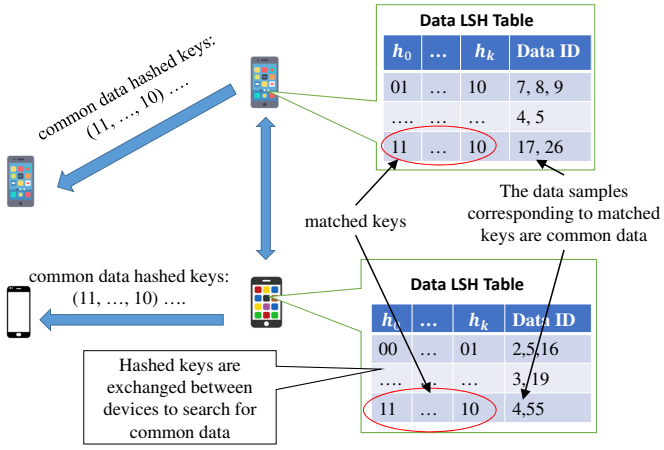

Fig. 2. Reducing Common Data with LSH in Page Control

on a mobile device can effectively contribute to the local model training even though it is identical to those on other devices. However, its contribution to global model training gradually diminishes after the local model gradients on the devices with similar data have been aggregated into the global model. The reason is that similar data tend to change local gradients in a similar way, and FedAvg is usually used in the aggregation. In one extreme case where two devices hold exactly the same local data, the aggregated gradients would be same as the gradients from any device's local training. Therefore, at the beginning of communication round, reducing common data among users can be an effective mechanism to accelerate training speed.

For federated learning, it is not a valid approach to collect data from users and then reduce common data. Thus, we propose to leverage the following advantages of LSH to detect and reduce common data. First, the LSH family hash function is practically infeasible to invert. Only exchanging LSH hash function values between users may not jeopardize the privacy in raw data. Second, the innate capability of LSH in clustering similar items makes it highly efficient in detecting common data.

As shown in Figure 2, each device can construct a LSH table for all local devices' data sets. In each round, a certain number of devices are randomly selected. They broadcast their hashed key vectors to each other, based on which common data is detected. Then, these devices broadcast the hashed key vectors of common data to all other devices, which mark their common data on their data hash table. The central server can configure faster devices to train more common data and adjust the size of training data for slow worker at next epoch.

Compared to dropping out neurons, dropping out common data may be more efficient and ideal when the model is trained on the massive and similar users' data with relatively short-length features for low communication overhead. Beside it, since data rarely change during the training, there is no need to update data hash tables repeatedly.

B. Pace Control via Split Learning

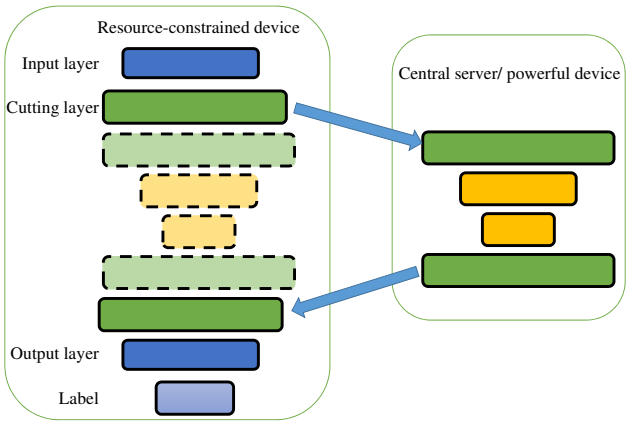

Fig. 3. Split Learning Configuration

Split learning [9] allows a device to offload the computation of some layers in its neural network to other devices. For privacy reasons, only the computation in middle layers are offloaded. Figure 3 shows that each client trains a partial network up to a specific cutting layer, the output would be sent to server/device for continue training, and eventually, server/device would send the input of end layer back to client in order to avoid label sharing. For pace control, a slow device may choose to offload the computation to the central server or to other participating devices that are more powerful. Compared to neuron dropout and common data dropout, split learning does not need to sacrifice the quality of data or model and no high cost to set up (e.g., building hash tables). However, because of the possible intensive information between cutting layers in neural network and the high frequency of data exchanging, the communication overhead can be considerable or even prohibitive. Therefore, split learning will be more suitable when the communication is not the bottleneck, i.e. a stable and resilient network being provided for all participants during whole process.

V. Summary and Conclusion

The pace-control mechanism presented in the paper dynamically adjusts the training speeds of mobile devices. It tunes up dropout rates for the mobile devices lagging behind to accelerate the training. It tunes down dropout rates for the mobile devices that train the model with high speed in order to improve model accuracy. At the same time, dropout is carried out to keep the neurons with high activation values. In classical back propagation algorithms, though not strictly proof, the connections between high activation neurons are more likely to produce higher gradients. Therefore, sparse gradient generated by the selected active neurons can effectively contribute to the convergence of the global model [10], [11].

This mechanism may bring three benefits to federated learning: 1) accelerating convergence; 2) improving quality of the final model; and 3) reducing the resource and energy consumption on mobile devices.

Compared to other solutions that rely on offloading data or workloads from stragglers to more power servers or peer devices, our mechanism achieves the above benefits while keeping user privacy to the largest extent.

References

[1] M. Rapp, R. Khalili, and J. Henkel, "Distributed learning on heterogeneous resource-constrained devices," *ArXiv e-prints*, 2020, cs.LG/2006.05403.

[2] A. Makhzani and B. Frey, "Winner-take-all autoencoders," *ArXiv e-prints*, 2015, cs.LG/1409.2752.

[3] A. Makhzani and B. Frey, "k-sparse autoencoders," *ArXiv e-prints*, 2014, cs.LG/1312.5663.

[4] R. Spring and A. Shrivastava, "Scalable and sustainable deep learning via randomized hashing," *ArXiv e-prints*, 2016, stat.ML/1602.08194.

[5] J. Gao, H. V. Jagadish, W. Lu, and B. C. Ooi, "Dsh: Data sensitive hashing for high-dimensional k-nnsearch," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, p. 1127–1138, 2014.

[6] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, (San Francisco, CA, USA), p. 518–529, Morgan Kaufmann Publishers Inc., 1999.

[7] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," *ArXiv e-prints*, 2016, cs.LG/1602.07868.

[8] B. Chen, T. Medini, J. Farwell, S. Gobriel, C. Tai, and A. Shrivastava, "Slide : In defense of smart algorithms over hardware acceleration for large-scale deep learning systems," *ArXiv e-prints*, 2020, cs.DC/1903.03129.

[9] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *ArXiv e-prints*, 2018, cs.LG/1812.00564.

[10] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017.

[11] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *ArXiv e-prints*, 2020, cs.CV/1712.01887.