Detailed comparison of communication efficiency of split learning and federated learning

Abhishek Singh, Praneeth Vepakomma, Otkrist Gupta, Ramesh Raskar
Massachusetts Institute of Technology
Cambridge, MA 02139
vepakom@mit.edu

Abstract

We compare communication efficiencies of two compelling distributed machine learning approaches of split learning and federated learning. We show useful settings under which each method outperforms the other in terms of communication efficiency. We consider various practical scenarios of distributed learning setup and juxtapose the two methods under various real-life scenarios. We consider settings of small and large number of clients as well as small models (1M - 6M parameters), large models (10M - 200M parameters) and very large models (1 Billion-100 Billion parameters). We show that increasing number of clients or increasing model size favors split learning setup over the federated while increasing the number of data samples while keeping the number of clients or model size low makes federated learning more communication efficient.

1 Introduction

Recent advances in deep learning has enabled ubiquitous applications in society and rapid growth of devices, data has ushered the need for distributed machine learning. Federated learning (McMahan et al., 2016; Konečný et al., 2016) and Split learning (Gupta & Raskar, 2018; Vepakomma et al., 2018, 2019) are two methods which allow to train a model collectively from various distributed data sources without sharing raw data. However, with such increasing number of devices (data sources) and increasing model complexity it is important to understand the role of these factors on communication efficiency of these distributed learning algorithms. In this work, we compare the communication efficiency of federated learning and split learning that allow training of deep neural networks from multiple data sources in a distributed fashion while not sharing the raw data in data sensitive applications.

Let us for example consider a network of data sources such as smart watches, hospitals, word corpus models or biobanks. Each of these entities have varying amounts of labeled data which we would like to use to train a deep learning pipeline. We ask ourselves on how one can train in a distributed setting without using too much communication bandwidth or computational burden at each of these locations?

We now describe split learning, federated learning in sections 1.1 and 1.2 and then compare in detail the communication efficiencies of both these approached in section 2 followed by further analysis of these efficiencies with various dataset sizes, increasing number of clients and model sizes in section 3.

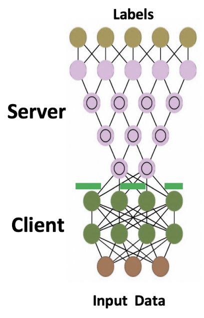


Figure 1: Vanilla split learning setup showing distribution of layers across client and server.

1.1 Split learning

The method for training split learning has two main parts, a.) the topology step and b.) the training step. The topology step involves dividing the neural network into two separate parts comprised of some layers in the beginning and remaining layers at the end as shown in Figure 1 . Both sides initialize network randomly and proceed to the following training steps.

The training steps involve forward propagation of data through the beginning layers by data source. In the simplest of configurations, the output tensor from these layers and the corresponding label is then transmitted over to the cloud. The cloud continues the forward propagation by processing the output tensor through its remaining layers. The cloud then computes gradients using the transmitted label and propagates the gradient backward. The gradient generated at the first layer of server is then transmitted back to client (data source) and these steps are repeated until convergence. By following these steps we can actually train the deep network without requiring sharing of raw data by client, or any details of the part of the model held by client or server. More advanced configurations of split learning where there is no label sharing or configurations for multi-task learning with vertically partitioned data, or multi-hop 'Tor' like communication are detailed in Vepakomma et al. (2018). In the case of multiple clients and one server, there are two approaches of training, one with weight synhcronization between any client i and the next client $i + 1$ after client i completes an epoch or across batches, and the other approach is without any client weight synchronization where clients take turns with alternating epochs in working with the server.

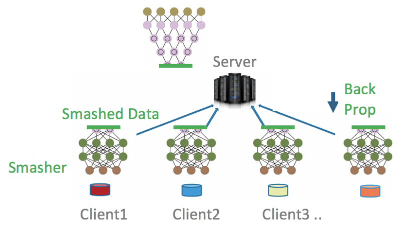


Figure 2: Split learning setup with multiple clients and a server with dotted green line showing the split between the client's share of layers and the server's share of layers. Activations from only the split layer (last layer of client) are shared during forward propagation and gradients from only first layer of server are shared with client during backpropagation.

1.2 Federated learning

In this other approach for distributed learning, every client runs a copy of the entire model on its own data. The server receives the weight updates from every client and averages them to get the updated weights from the server. The updated weights are then downloaded by the clients and the process continues until convergence.

2 Communication efficiency

In this section we describe our calculations of the communication efficiency for both of the distributed learning setups of split learning and federated learning. For analyzing the communication efficiency, we consider the amount of data transferred by every client for the training and client weight synchronization since rest of the factors affecting the communication rate is dependent on the setup of training cluster and is independent of the distributed learning setup. We use the following notation to mathematically measure the communication efficiencies,

Notation: K = # clients, N = # model parameters, p = total dataset size, q = size of the smashed layer, η = fraction of model parameters (weights) with client and therefore $1 - \eta$ is fraction of parameters with server.

In Table 1 we show the communication required per client per one epoch as well as total communication required across all clients per one epoch. As there are K clients, when size of the training dataset

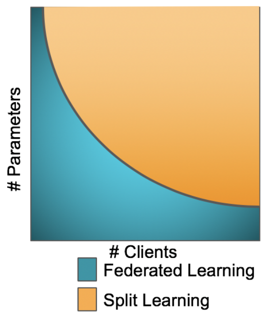


Figure 3: Hyperbola dividing the regions where one technique perfors better over the other and both the feasible regions are shown in this theoretical schematic cartoon. Real instances of this equation are shown in the Analysis section.

across each client is the same, there would be p/K data records per client in split learning. Therefore during forward propagation the size of the activations that are communicated per client in split learning is $(p/K)q$ and during backward propagation the size of gradients communicated per client is also $(p/K)q$. In the case where there is client weight sharing, passing on the weights to next client would involve a communication of ηN.

In federated learning the communication of weights/gradients during upload of individual client weights and download of averaged weights are both of size N each. The split learning setup has

Method	Communication per client	Total communication
Split learning with client weight sharing	$(p/K)q + (p/K)q + \eta N$	$2pq + \eta NK$
Split learning with no client weight sharing	$(p/K)q + (p/K)q$	$2pq$
Federated learning	$2N$	$2KN$

Table 1: Communication per client and total communication for the distributed learning setup as measured by the data transferred by all of the nodes in the learning setup.

two variants where one involves weight sharing among clients while the other variant does not. Sharing weights among clients helps in the synchronization among the clients but at the same time leaks more information as held by the weights of the model. Weight sharing among client adds extra communication overhead where the amount of overhead depends upon the model size (N) and the size of the smashed layer (q). The variant that does not involve weight sharing is based on alternating turns of epochs taken by the clients in working with the server. The communication efficiency computed as ratio of data transfers of federated learning and split learning therefore is given by $\rho = \frac{2NK}{2pq+\eta NK}$. Split learning wins in terems of communication efficiency in scenarios when $\rho > 1$ and federated learning wins when $\rho < 1$. Upon rearranging the terms, and expressing it as an equality, and dividing the numerator and denominator by NK we get the equation of a rectangular hyperbola as $N = \frac{2pq}{((2-\eta)K)}$ in the case of client weight sharing and $N = \frac{pq}{K}$ in the case of no client weight sharing with alternating epochs. This hyperbola divides the regions where one technique perfors better over the other and both the feasible regions are shown in the theoretical schematic cartoon in Figure 3.

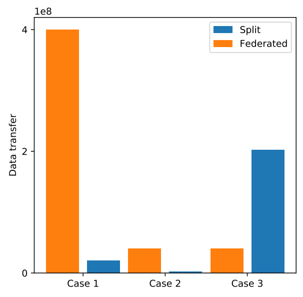


Figure 4: Use case corresponding to smart watches. Case-1 refers to a big model and high number of client setting. Case-2 refers to a relatively small model and Case-3 refers to even smaller model as well as low number of parameters.

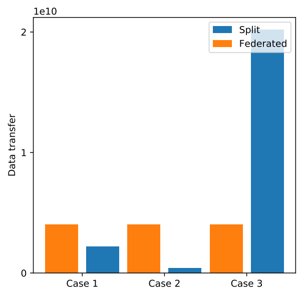


Figure 5: Use case for the healthcare scenario. Case-1 refers to a big model size and small number of clients. Case-2 has slightly higher number of clients and rest everything is same. Case-3 has bigger dataset and less number of clients.

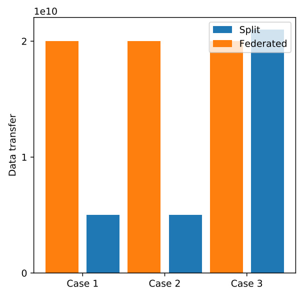
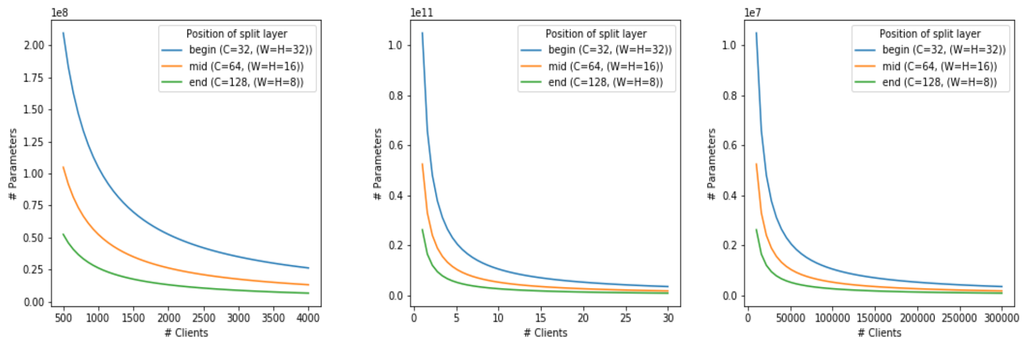


Figure 6: Amount of data transfer required for completing one round of training is shown for both split learning and federated learning like in adjacent figures and lower value on the y-axis means higher communication efficiency.

3 Analysis

We consider some of the real-life use cases which depict the relative efficacy of the two methods of split learning and federated learning.

First case is motivated by the scenario where the number of clients is in the range of millions. One concrete use case is smart watches (edge devices) which comprises of users in a diverse range from hundreds to millions. Figure(4) presents all the three cases for the training under three different scenarios. Starting from the case-1, split learning setup has relatively lower data transfer due to the high number of model parameters and high number of clients. As we reduce the model parameters and number of clients in the case-3, the federated learning setup is relatively more communication efficient. Second case, Figure(5) is inspired from the healthcare setting where the clients are hospitals with large models but the number of clients is limited. In this case federated learning and split learning perform roughly same except the case three where federated performs better when dataset size is bigger and the number of clients is small.
Third case, Figure(6) covers the use case for bigger institutions like biobanks where all the three parameters are in a high number. In case-1 and case-2, split learning setup outperforms the distributed learning because it scales well with the number of clients and the model parameters.
Figure(7) provides more practical scenarios of feasible curve of effeciency across a broad spectrum of parameters.



(a) Number of clients in a setting where the number of parameters for the model is in the range of $10M - 200M$.

(b) Small client setting where massive models can be fit like Transformers and AmoebaNet(Huang et al. (2018)).

(c) Large client setting which allows a big range from tiny to large models to fit in is shown in this subfigure.

Figure 7: Curve for comparing effeciency of Split learning and Federated learning setup. Similar to the figure 3, upper half region shows the feasible region for relatively higher communication efficiency for Split Learning. The three curves in all three figures refer to the change in the position of split layer. We consider the size of commonly used activations in the CNN models during early layers.

4 Conclusion and future work

Our analysis suggests that the split learning setup becomes more communication efficient with increasing number of clients and is highly scalable with number of model parameters. Federated learning becomes more efficient with increasing the number of data samples especially when the number of clients is small or model size is small. In this work we also identify some of the use-cases where one would be more effective than the other in terms of communication efficiency. The analysis and discussion presented in this work would be benefecial for the distributed learning community to understand the potential benefits of both methods. This work could be extended by analyzing the resource utilization and number of epochs required for convergance of both distributed learning setups under different practical scenarios. (Gupta & Raskar, 2018; Vepakomma et al., 2018) shows that split learning converges drastically quicker than federated learning.

References

Gupta, O. and Raskar, R. Distributed learning of deep neural network over multiple agents. *CoRR*, abs/1810.06060, 2018. URL `http://arxiv.org/abs/1810.06060`.

Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q. V., and Chen, Z. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *CoRR*, abs/1811.06965, 2018. URL `http://arxiv.org/abs/1811.06965`.

Konečnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

McMahan, H. B., Moore, E., Ramage, D., and y Arcas, B. A. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016. URL `http://arxiv.org/abs/1602.05629`.

Vepakomma, P., Gupta, O., Swedish, T., and Raskar, R. Split learning for health: Distributed deep learning without sharing raw patient data. *CoRR*, abs/1812.00564, 2018. URL `http://arxiv.org/abs/1812.00564`.

Vepakomma, P., Gupta, O., Dubey, A., and Raskar, R. Reducing leakage in distributed deep learning for sensitive health data. *arXiv preprint arXiv:1812.00564*, 2019.