Scalable Machine Learning ID2223 Project: An Investigation Into Split Learning

Xenia Ioannidou - ppio@kth.se **Bas T. Straathof -** btstr@kth.se

January 12, 2020

1 Introduction

This report communicates findings from experiments with a novel approach to decentralized distributed learning called split learning (SL), or *splitNN* [5, 14]. The motivation of SL is that data is often distributed over various entities that do not allow raw data sharing. Collaboration would be beneficial for all entities involved, however, a lack of trust, strict data sharing policies and limited customer consent usually impedes this. SL allows collaboration of multiple entities without raw data sharing and without significantly hampering model performance. Plus, it requires less computational resources and communication bandwidth than previously proposed methods such as federated learning (FL) [9] and large batch synchronous stochastic gradient descent (LBS-SGD) [1]. This report offers a brief theoretical overview of SL, as well as a proof-of-concept comparison against FL and regular non-distributed learning (RL). Afterwards the results will be analyzed and several limitations of the experiment and technical differences between SL and FL will be discussed.

2 Theoretical Overview

Split Learning Split learning comes in multiple capacities, but the scope of this project confines it to the most general case, which is relatively simple. In vanilla split learning, each of n data containing entities consecutively performs a forward pass on a mini-batch through a partial neural network, up to the cut layer. The weight matrix at the cut layer is sent to a central server, at which the forward pass is continued to the end of the network. Subsequently, back-propagation is performed up to the cut layer. The gradient update of the first layer on the server is sent back to all the clients, which complete the back-propagation up to their input data. This process can be repeated until the network is trained completely, in a way which leaves the central server agnostic w.r.t. to the raw input data. Note how each of the n partial networks up to and including the split layer are all slightly different, but that there is one unique version of the rest of the network on the server.

Federated Learning It being the more mature technology, federated learning also comes in various capacities. In its simplest form, federated learning trains n copies of a network at n distinct data containing entities. After a single forward and backward pass through the model at each entity, the weight matrices of all layers are sent to a central compute server which averages all weight matrices using the `FederatedAveragingh` algorithm[9].

Large Batch Synchronous Stochastic Gradient Descent[1] Another approach to distributed training on large scale data sets is Large Batch Synchronous Stochastic Gradient Descent (LBS-SGD) [1]. In LBS-SGD, a set of workers communicates with a set of servers in a synchronous way to eliminate stale gradients. Each worker updates its gradients with respect to outdated parameters on the server, hence in the LBS-SGD approach, each server only updates its parameters when all workers have sent their gradients. The servers aggregate and update the gradients and send them to all workers.

[1] Due to our colliding agendas during the holiday period, the task was split up as follows: Bas implementing SL and FL, and Polyxeni implementing LBS-SGD. As of yet, the LBS-SGD implementation has not been finished, but Polyxeni will try to have a working implementation ready at the time of the presentation.

Additional backup workers are used to wrap around the problem of waiting for a long time due to the slowest workers. This means that when the parameter servers receive gradients from any N workers, they make the appropriate updates of their parameters by using all N gradients.

3 Experiments

The main experiment described here revolves on a simple proof of concept to compare SL against FL. The main purpose of the experiment is to get a feel for the core concepts of both methods, and to build a scaffolding for discussing and comparing their strengths and weaknesses. Before outlining the methodology, the data set and network architecture are described. The results of the experiments and the analysis are reported in a separate sub-section.

3.1 Data Set

Due to a limited scope regarding time and computational resources, a relatively small benchmark data set had to be chosen. The data set of choice was `Fashion-MNIST` [16], a data set comprising of 70,000 images of fashion items from 10 categories, with 7,000 images per category. The images are 28 x 28 pixels and grayscale. The training set contains 60,000 images and the test set 10,000. `Fashion-MNIST` seems like an apt choice, since the task of classifying regular `MNIST` is too easy to solve with Convolutional Neural Networks (CNNs) [16].

3.2 Network Architecture

The choice of a network was largely dependent on the available computational resources. A CNN based on the LeNet-5 architecture [7], the only differences being that ReLU activation functions are used instead of tanh activation functions.

3.3 Methodology

For both SL and FL, the `FashionMNIST` training set is partitioned in 10 batches of 7,000 images each, which are allocated to 10 different clients. To ensure a fair comparison, the hyper-parameters are set equally across the tasks. As training optimizer, stochastic gradient descent is used with a learning rate of 0.01. Early stopping was adopted such that the models could not overfit on the training data. The batch size for training is 64, and the one for testing 1,000. Both SL and FL are implemented using the `PyTorch` v1.3.0 deep learning framework [11]. Now, the method specific implementation details and the evaluation methhod are explained.

SL implementation To the best of our knowledge, there is no publicly available implementation of SL. The authors of the original paper [15] were so kind to share a minimal code example of SL using the Message Passing Interface (MIP) [4]. The implementation here described also makes use of `mpi4py` [2]. Using MPI software n processes can be started simultaneously with the `mpirun` command. In this case, `mpirun -n 11 python3.7 split_learning.py` runs the `split_learning.py` 11 times. Inside the script, `mpirun` is leveraged to define the server and the 10 entities, which can communicate by sending messages. For the purposes of the experiment, each of the 10 clients contains the first convolutional block of the LeNet-5 CNN architecture, and the server contains the second convolutional block and the fully-connected block. Whereas it is currently not possible to use momentum and weight decay in FL[8], this is no issue for SL, hence, they are used and set to 0.9 and $5 * 10^{-4}$, respectively.

FL implementation A popular open source Python library for FL is `PySyft` [12]. `PySyft` efficiently takes care of communication between the entities and the server, as well as the `FederatedAveraging` algorithm.

Evaluation In the implementations of SL and FL, the training procedure is continued until the loss on the test set converges or worsens. Since in FL all entity use the same model, evaluating the test set is straight forward. In split learning, there is one unique partial model after the split learning, but each entity has its own version of the partial version of the first convolutional block. This complicates

evaluation, but the most straight-forward choice is to evaluate on the model of the last entity, since it uses the partial model on the server that has been trained on the data of all other entities.

3.4 Results

All experiments were carried out on the same machine with a 4-core CPU. It was observed that for this experimental setup, SL and RL converged after 10 epochs whereas FL required 25 epochs. The training times were 4 minutes and 22 seconds (avg. 26.2s per epoch) for SL, 21 minutes and 17 seconds (avg. 51.5s per epoch) for FL, and 3 minutes and 20 seconds (avg. 19.0s per epoch) for RL. The test loss and accuracy curves are displayed in Figure 1. It can thus be determined that in this scenario SL is faster by a factor of two, but that the methods perform equally well if granted enough time. SL is only a factor of 1.3 slower than RL.

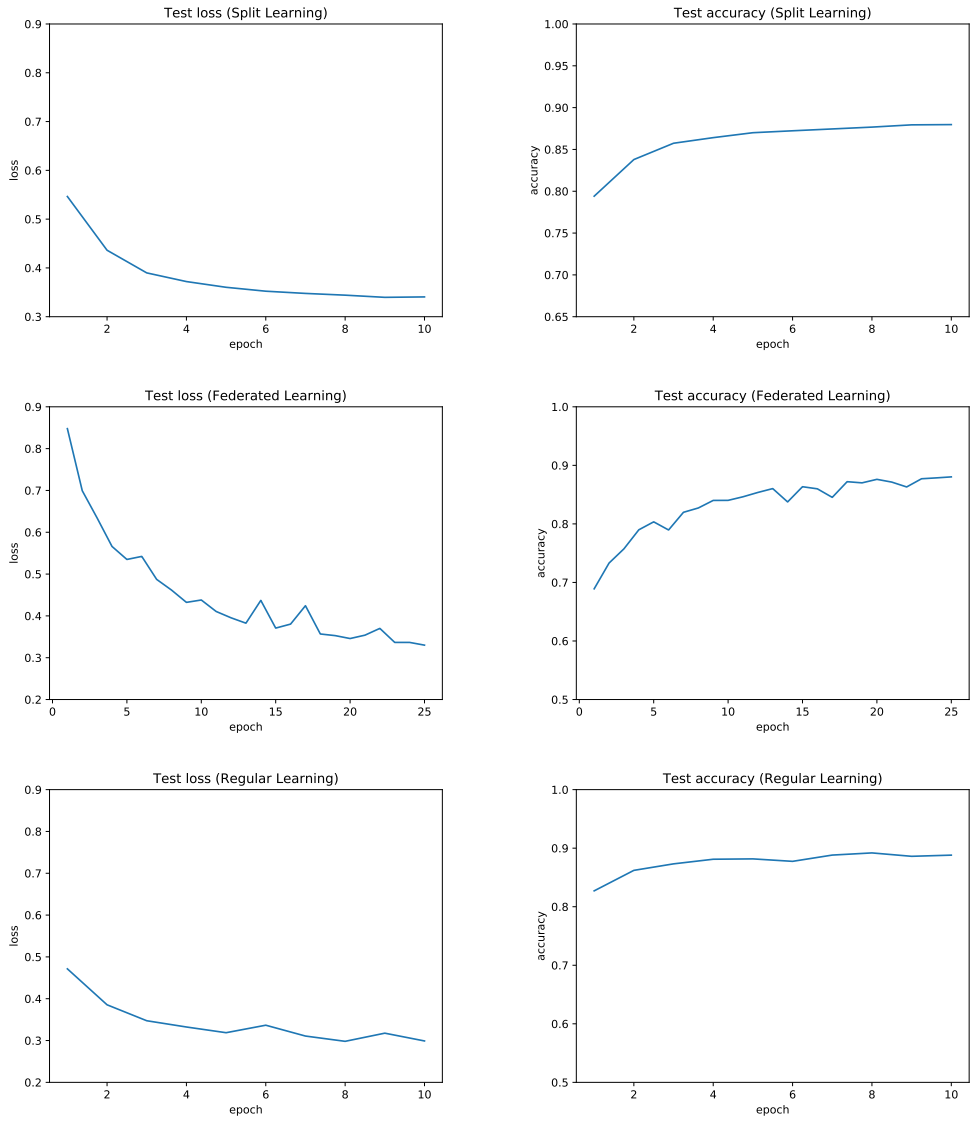

Figure 1: (upper left) SL test loss curve (min. $\tilde{0}.34$). (upper right) SL test accuracy curve (max $\tilde{8}8.0\%$). (center left) FL test loss curve (min. $\tilde{0}.33$). (center right) FL test accuracy curve (max $\tilde{8}8.0\%$). (lower left) Standard learning test loss curve (min. $\tilde{0}.30$). (lower right) Standard learning test accuracy curve (max $\tilde{8}8.8\%$).

4 Advantages and Limitations

It should be stressed that this project is mainly exploratory, and that the experiments conducted are of insufficient potency to draw significant results. Here, the methods will be compared on four aspects: performance, privacy, computational resources and bandwidth.

Performance This experiment demonstrates that SL and FL can perform equally well, but that it takes FL twice as long to converge and that both are slower than regular learning and perform less well. This result might not be generalizable, as previous work has reported that SL has a significant performance advantage over FL [14]. To be conclusive, more rigorous experiments need to be carried out. In its current form, however, FL suffers from data that is distributed in a non-IID way over the different entities [6], whereas this should not hinder SL. Moreover, in SL momentum, weight decay and batch normalization can be used, whereas this might not be possible in FL [8], which also provides a performance edge for SL.

Privacy SL and FL are both devised to enhance data privacy. For FL and SL with a single layer at the entity, however, by using the updated weights private information can still be accessed if a certain entity for example only contains a single record. FL being the more mature technology, it has been demonstrated that it is compatible with differential privacy (DF) [3, 10] and secured multi-party communication (SMPC) encryption technologies, which mitigate this effect. It is left to be studied whether such technologies are compatible with SL. Nevertheless, it has been proven that an auxiliary loss function can be employed to maximize the Kullback-Leibler divergence between the weight matrix of the cut layer in SL and the raw input data [15], which is said to enhance the privacy. Moreover, the more layers are placed at the entity, the safer SL seems to become, however, this might render it less performing.

Computational Resources From this experiment, it seems that twice as much computational power is required for FL to perform as well as SL. However, this may be dependable on various factors such as the size of the data set, the network architecture and the location of the cut layer. Future research has to investigate these properties. Nevertheless, in the case that the entities have limited computational resources but the central server being potent, SL has a clear advantage over FL.

Bandwidth According to a recent paper, there is a clear cut formula to determine whether FL or SL is more communication efficient [13] depending on the number of client entities, the number of model parameters, the total data size, the size of the smashed layer and the fraction of model parameters that are on the client.

5 Conclusion

This reports shows that SL and FL are comparable technologies in a simple scenario, but that it should be investigated in a case-by-case manner which technology to use to solve a specific problem. Moreover, it indicates that both technology are still in their infancy, and that more research needs to be conducted to see how safe scalable and safe they are. Lastly, it should be studied how both techniques relate to regular, non-distributed learning in terms of performance, computational resources and bandwidth.

References

[1] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[2] Lisandro D Dalcin, Rodrigo R Paz, Pablo A Kler, and Alejandro Cosimo. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124–1139, 2011.

[3] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[4] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.

[5] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.

[6] Heejae Kim, Taewoo Kim, and Chan-Hyun Youn. On federated learning of deep networks from non-iid data: paramter divergence and the effects of hypterparametric methods. *Under review at ILCLR 2020 Converence*, 2020.

[7] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[8] Wei Liu, Li Chen, Yunfei Chen, and Wenyi Zhang. Accelerating federated learning via momentum gradient descent. *arXiv preprint arXiv:1910.03197*, 2019.

[9] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[10] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963*, 2017.

[11] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[12] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*, 2018.

[13] Abhishek Singh, Praneeth Vepakomma, Otkrist Gupta, and Ramesh Raskar. Detailed comparison of communication efficiency of split learning and federated learning. *arXiv preprint arXiv:1909.09145*, 2019.

[14] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

[15] Praneeth Vepakomma, Otkrist Gupta, Abhimanyu Dubey, and Ramesh Raskar. Reducing leakage in distributed deep learning for sensitive health data. *arXiv preprint arXiv:1812.00564*, 2019.

[16] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.