2020 International Conference on Information Technology Systems and Innovation (ICITSI)
Bandung - Padang, October 19 - 23, 2020
ISBN: 978-1-7281-8196-7

Study of Data Imbalance and Asynchronous Aggregation Algorithm on Federated Learning System

Senapati Sang Diwangkara
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
13516107@std.stei.itb.ac.id

Achmad Imam Kistijantoro
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
imam@stei.itb.ac.id

***Abstract*—As the use of machine learning techniques are becoming more widespread, the need for more elaborate dataset is becoming more prevalent. This is usually done with data collection methods that pay little to no attention to the data owner's privacy and consent. Federated learning is an approach that tries to solve this problem, where such system can train a machine learning model without centrally storing the needed data. But one weakness of the current implementation is that they have a slow convergence time, despite the fact that they distribute the task on many nodes. This is mainly caused by the synchronous nature of the current algorithm. In this paper, we observe the effect that asynchronous aggregation algorithm has on convergence time and test the two factors that might affect it – staleness and data imbalance – on various levels. We implement the asynchronous aggregation algorithm by adapting the Stale Synchronous Parallel algorithm. We test our system on MNIST dataset and found that asynchronous aggregation algorithm improves convergence time in a federated learning system that has large inequality in server-wise update frequency and has a relatively balanced data distribution.**

***Index Terms*—asynchronous, non-iid, federated learning, machine learning, distributed system, distributed training**

I. Introduction

In an era where machine learning algorithm is being utilised in many significant projects, collecting a suitable dataset for said project is important to guarantee a good model performance. Fortunately, this has been made possible by the widespread use of social media and smartphone, that enables the intelligence behind GBoard [19] and Google Maps [20] among other things.

But more often than not, the data used in those kind of scenarios is personal in nature, which would disturb many users if not collected and handled properly. Unfortunately, incidents regarding improper handling of personal data is relatively rampant [22] [21] [23]. Because of this, the awareness and need for a mechanism to properly control data access is becoming more relevant. One of many techniques that want to counter this trend is Federated Learning.

Federated Learning is a learning technique that allows users to collectively reap the benefits of shared models trained from a rich, mobile data, without the need to centrally store it [12]. This technique would allow the system to solve a Federated Optimization problem, in which a model owner (client) would distribute their model to each data owner (server) for it to be trained and then gather and aggregate the gradients using federated aggregation algorithm [12] to iteratively update the model.

While Federated Learning itself doesn't necessarily guarantee data privacy, it is the backbone in which complementary technologies like Secure Multi-Party Computation, Homomorphic Encryption, and Differential Privacy techniques can be implemented on [13]. This paper however, mainly focus on Federated Learning itself.

We can view Federated Optimization more holistically by contrasting it with a similar and more popular technique, which is Distributed Optimization. We can model Federated Optimization as a special case of Distributed Optimization that has a few properties [12]:

- Non-IID: The generated data is typically based on the use case of a particular node, and will not be representative of the population distribution.
- Imbalanced data size: Similarly, those different use cases may lead to varying amounts of local training data.
- Massively distributed: The number of nodes participating in an optimization is anticipated to be much larger than the average number of examples per node.
- Limited communication: Connections are often slow, expensive, and/or lossy.

Currently, Federated Learning system has been used in many IoT projects [14] [18] and mobile app [6]. However, it is still an actively researched topic as it is fairly new, and as such, still have many things to be desired. In this work, we emphasis on the problem of convergence time [2], that is, Federated Learning system is significantly slower than its centralized counterparts, despite the fact that the optimization task are distributed into many workers. In order to give light to this issue, we will investigate one particular weakness in the current architecture, which is its synchronous nature.

In the current architecture, a node must wait for every other node to finish their current training round before it is allowed advance into the next training round. This is an inefficient utilization of the computing power, as the server spent most of their time waiting for the next instruction rather than training. This is a problem that asynchronous algorithm would solve. An aggregation algorithm is said to be asynchronous if it doesn't require its workers to be in the same round at any given time. This will give way to a more efficient resource utilization and possibly faster optimization performance, although not with its own caveat.

By adopting asynchrony, we relax the implicit requirement of model consistency, that the model being used for training must be the same as the model being updated. This means that an update sent by a server might be outdated, thus not a very optimal gradient, because the client's model is being updated by another server in the meantime. Nevertheless, this doesn't give much negative effect to distributed learning system's optimization performance, as shown by many successful asynchronous system in distributed learning [1].

However, we made a hypothesis that model consistency would play a bigger role in a federated learning system because of its 2 defining characteristics: Non-IID data and network limitation. Our reasoning is as follows:

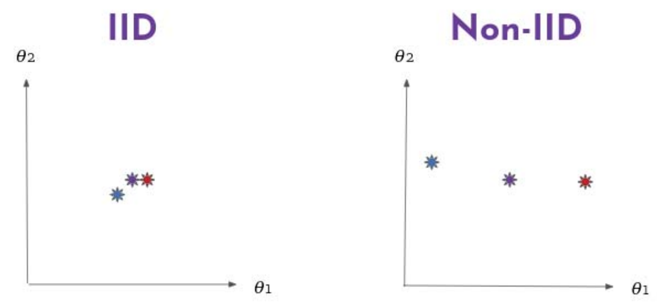

Fig. 1. Ilustration of the parameter plane of a system with 2 servers. Purple is the optimal point when data is centralized, blue and red is the optimal point according to server 1 and 2 respectively.

Any particular server in a distributed training system that has a non-IID dataset distribution would have a local dataset that isn't representative of the overall dataset population. Hence, a model fitted for the data in a particular server would be very different than the ideal model, i.e. the fitted model if all data is stored centrally. It would also be very different from each other, or said metaphorically, each server would have a wildly different "opinion" of what the ideal model should look like. Hence, we hypothesize that in a non-IID setting, an update sent from each server might hurt the client's model performance if not applied with equal frequency.

Unfortunately, Federated Learning's network characteristics may cause unequal update frequency to happen, as one server might have a slower and/or more lossy connection to the client than another server.

Because of those theoretical downside, in this work, we investigate the behavior of synchronous, semi-synchronous, and asynchronous aggregation algorithm in a federated learning system. Our contribution goal is to determine whether asynchronous algorithm will always have a positive impact on optimization performance or whether it requires a certain condition to achieve it.

We publish the code of our work in https://github.com/diwangs/asynchronous-federated-learning

II. Related Work

A. Model Consistecy in Distributed Learning

Many advances has been made in the field of distributed learning for the past few years. In Ben-Nun et. al. meta-analysis paper [1], this advances include many techniques of concurrency in training. One of the aspect of training concurrency that is relevant for this work is the model consolidation strategy, i.e. how is the model combined in-between training. This is defined in terms of the degree of model consistency between each node, ranging from from synchronous learning [4] [16], where the models is combined every round resulting to a consistent model, to ensemble learning [8] [10] [15], where model is never combined.

One approach that is in the middle of these two extremes is the stale-synchronous approach [5] [17] [7]. Stale-synchronous aggregation algorithm introduces asynchrony while still maintaining some consistency guarantee.

B. Federated Learning

Federated learning approach itself is first proposed by McMahan et. al. [12], in which they propose the Federated Averaging algorithm that underlies Federated Learning techniques. This algorithm has been used and evolved into many subsequent applications [14] [18] [6] and system design [2]

Many authors has also introduced asynchrony into the original algorithm [11] [3] in order to ramp up optimization performance. To combat staleness, these algorithm often incorporate temporal weighting, a mechanism that will scale the gradient updates smaller as the staleness gets bigger. However, as far as the authors are aware, none of these studies incorporates the effect of data imbalance on optimization performance.

III. Experiment Configuration

A. Framework

In this experiment, we will use PySyft [13] as a base framework to build a federated learning system. PySyft is a Python library that provides a client and server class, connected via WebSocket, that can facilitate federated optimization for model built on popular deep learning framework such as TensorFlow and PyTorch. We use PyTorch for the deep learning framework, because at the time of experiment, PyTorch is the most maintained system for PySyft.

The experiment will consists of many server (data owner) processes and 1 client (model owner) process with many threads, each connecting to 1 server process. This combination of multiprocessing and multithreading is done to get around cpython's GIL limitation, as we want each server to run in parallel. The client would also has a separate component to evaluate in-training model periodically. The schematics of this system can be examined at Fig. 2.

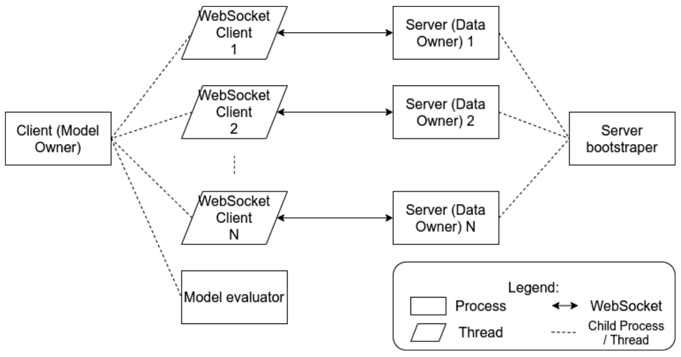

Fig. 2. Overall experiment architecture

For the actual workload of the experiment, we use an image classification task with MNIST dataset [9] on a total of 4 server processes that runs on the same machine. We will use Stochastic Gradient Descend (SGD) optimizer, as at the time of experiment, PySyft doesn't properly support more advanced momentum-based optimizer. We use the default learning rate value for SGD, which is 0.1.

B. Network Limitation

To simulate network limitation, we use traffic control (tc) utility on Linux to add a modifier such as loss and delay to a network interface, specifically using tc's network emulator (netem) module. We limit ourselves to only investigate the effect of globally limited network i.e. the netem modifier will be applied to all node. The effect of more elaborate, node-specific limitation can be investigated further on future work.

In this experiment we only tweak the rate of random network loss i.e. random packet drop rate (`L`) as a representation for network limitation, because we can represent both network loss and slowness with only one configuration, as the communication protocol used by PySyft, WebSocket and TCP, both features timeout resend mechanism. We test 3 `L` configuration: 0%, to represent ideal network, 10% to represent TCP timeout, and 20% to represent the default WebSocket timeout (60 s), as in preliminary test, minor difference in `L` will only result in negligible difference.

C. non-IID Data

To simulate non-IID data, we've created a custom data splitter. For each class in the dataset we will randomly pick data from that class for each server with an amount determined by a normal distribution, whose mean is the average amount a server should get (the amount of data in that class divided by the amount of server) and whose standard deviation is a configurable hyperparameter (`D`). This variable represents a the degree of data imbalance the system would have, as the data will be evenly distributed when `D` is 0 and when `D` is large, the dataset distribution will be very skewed.

We test 3 `D` configuration: 0, to represent even distribution, 600, to represent mild imbalance, and 10000, to represent severe imbalance in which a server might not have a certain class.

D. Asynchrony

We adapt the Stale Synchronous Parallel (SSP) [7] algorithm to be used in this experiment. We choose it, instead of temporal weighting mechanism in other Asynchronous Federated Learning system [11] [3], because of its ability to abstract synchronous, semi-synchronous, and asynchronous aggregation algorithm into just one algorithm with different hyperparameter. This algorithm introduces a staleness threshold (`S`) hyperparameter that limits the maximum distance of round any pair of node might have. Training starts with asynchronous algorithm, but if there's a node that crossed the threshold, that node will be forced to wait until the threshold advances. Synchronous algorithm will be represented when `S` is 0 and asynchronous algorithm will be represented when `S` is infinite.

To accommodate this algorithm, we will also modify the FedAvg [12] algorithm to be able to be used in an asynchronous setting. Instead of waiting for all gradients to be collected and then averaging them before applying them to update our model, we apply the recently received gradient as soon as it arrived by first weighting them appropriately to match the average formula.

$$model+ = 1/nserver * gradient$$

We test 4 `S` configuration: 0, to test synchronous algorithm; infinite, to test asynchronous algorithm; 6 and 12, to test semi-synchronous algorithm, because at the initial testing, we found that the maximum staleness achieved by asynchronous system is 16, so we pick 2 middle points to represent semi-synchronous.

E. Hardware and Software Environment

We run the experiment on a computer with 64-core Intel® Xeon® E5-4610 CPU running at 2.3 GHz and 128 GB of RAM. For this experiment, we use PySyft 0.2.3 running on Python 3.8.

To test our hypothesis, we will periodically measure the model performance (using F1 metric) every 15 seconds to asses the system's optimization performance. The experiment is repeated 3 times for each configuration pair, each run for 40 minutes.

IV. Result and Analysis

In this section we will talk about the key results of our experiment.

We can see the effect of data imbalance in Fig. 3. As shown in the graph, we found that when `L` stays the same, as `D` gets larger, the system with larger `S` has worse performance than the system with small `S`. In this experiment, this manifested as optimization instability, where the model performance will occasionally dips deep down before bounces back up again.

We can see the effect of network imperfection in Fig. 4. As shown in the graph, we found that the graph of `L = 0%` and `L = 10%` looks relatively the same as opposed to `L = 20%`, which is less "scribbled" as it noticably has less update

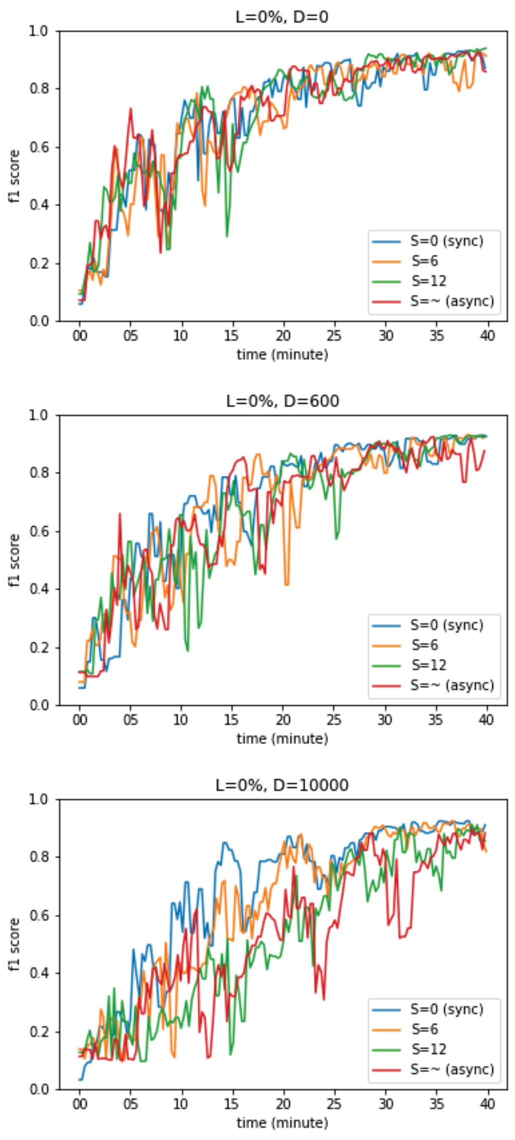

Fig. 3. The effect of data imbalance on optimization performance

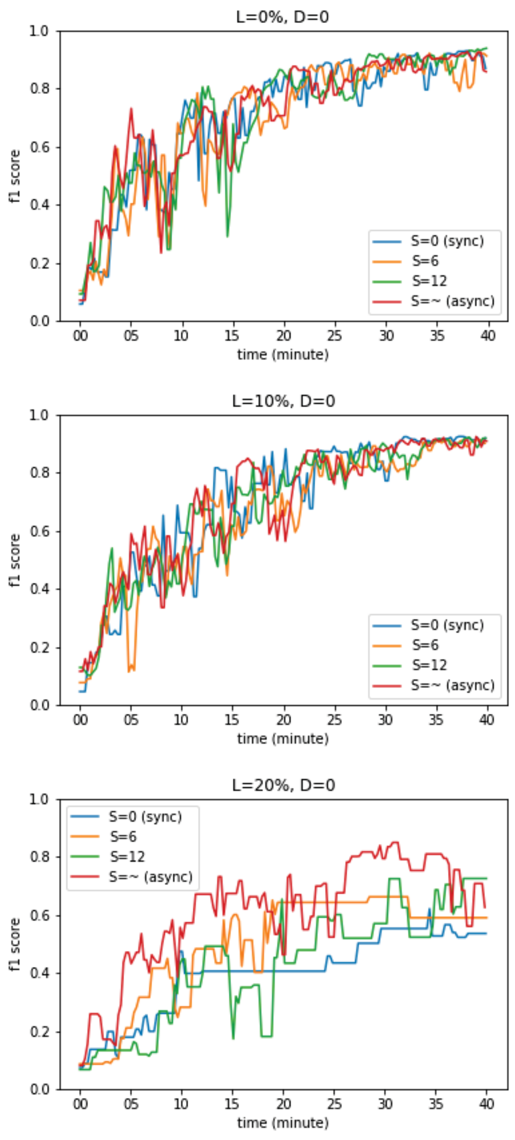

Fig. 4. The effect of network loss on optimization performance

than the former and have more straight line in the model performance graph, as such system spend much time waiting on a stale node to catch up.

The combined effect of both data imbalance and network imperfection is shown in Fig. 5. We can examine the combined effect of large L and large D when we look at L = 20% model performance graph. At L = 20%, when D = 0 and D = 600, system with large S has a better optimization performance than system with small S.

Interesting finding is found when we compare 3 runs of lossy network and imbalance dataset configuration. As shown in Fig. 6, when L = 20 and D = 10000, 2 out of 3 runs show that all system has similar optimization performance. Only 1 exception run shows that system with large S has better optimization performance than system with small S.

These findings fail to falsify our previously stated hypothesis where in a non-IID setting, an update sent from each server might hurt the client's model performance if not applied with equal frequency. These findings also tells us that we can equalize the update frequency by limiting the asynchrony of the system at the cost of the frequency itself.

V. Conclusion

We conclude that, compared to synchronous aggregation algorithm, asynchronous aggregation algorithm in a federated learning system improves the optimization performance of a system that has large inequality in server-wise update frequency and relatively balanced data distribution.

Beside that, it may improve the optimization performance of a system that has large inequality in server-wise update frequency and relatively imbalanced data distribution, depends on the randomness of the aggregation.

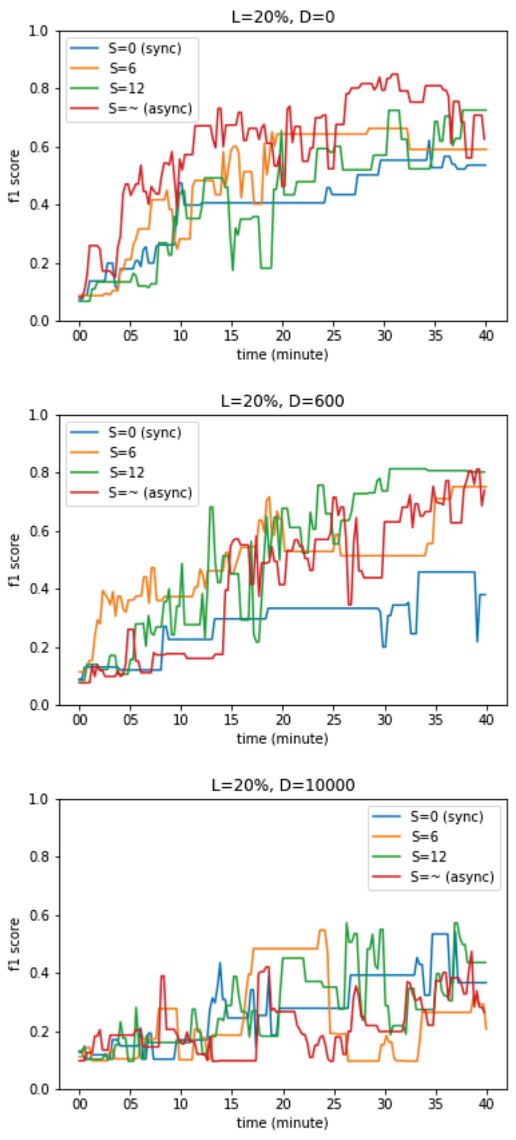

Fig. 5. The effect of data imbalance and network loss on optimization performance

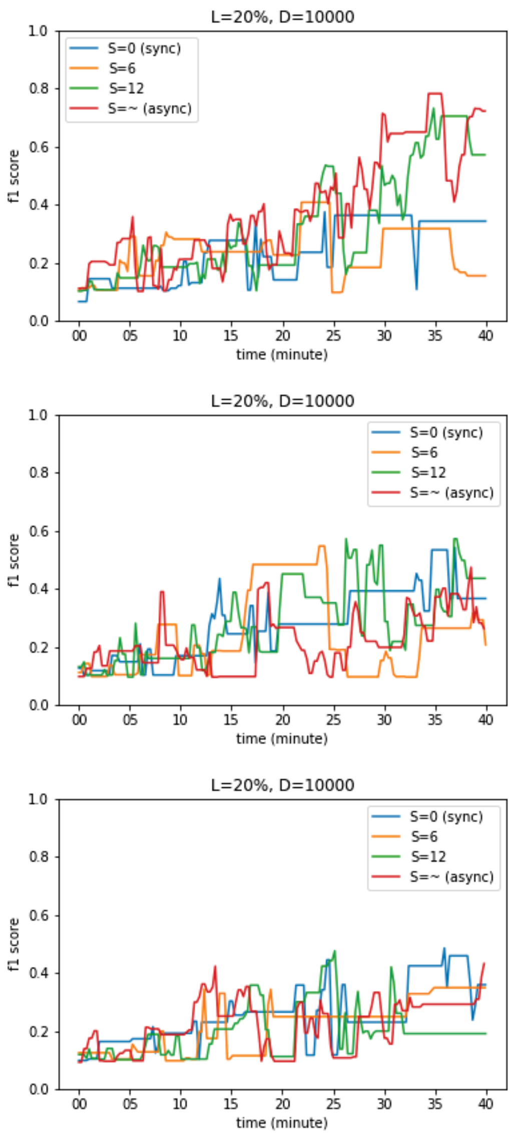

Fig. 6. 3 different runs on very lossy network and highly imbalance data

However, it worsen the optimization performance of a system that has small inequality in server-wise update frequency and relatively imbalanced data distribution.

Moreover, it has no effect on the system that has small inequality in server-wise update frequency and relatively balanced data distribution.

VI. Future Work

We can take this work further in various direction, including, but not limited to:

- Testing with more hyperparameter configuration values
- Adding new hyperparameter such as network delay and per-node configuration
- Exploring other semi-synchronous algorithm other than SSP
- Evaluating the use of GPU, as it may also contributes to the update frequency inequality

References

[1] Ben-Nun, Tal, and Torsten Hoefler. "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis." ACM Computing Surveys (CSUR) 52.4 (2019): 65.

[2] Bonawitz, Keith, et al. "Towards federated learning at scale: System design." arXiv preprint arXiv:1902.01046 (2019)

[3] Chen, Yang, Xiaoyan Sun, and Yaochu Jin. "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation." IEEE Transactions on Neural Networks and Learning Systems (2019).

[4] Coates, Adam, et al. "Deep learning with COTS HPC systems." International conference on machine learning. 2013.

[5] Gupta, Suyog, Wei Zhang, and Fei Wang. "Model Accuracy and Runtime Tradeoff in Distributed Deep Learning: A Systematic Study." arXiv preprint arXiv:1509.04210 (2015).

[6] Hard, Andrew, et al. "Federated learning for mobile keyboard prediction." arXiv preprint arXiv:1811.03604 (2018).

[7] Ho, Qirong, et al. ”More effective distributed ml via a stale synchronous parallel parameter server.” Advances in neural information processing systems. 2013
[8] Im, Daniel Jiwoong, et al. ”Generative adversarial parallelization.” arXiv preprint arXiv:1612.04021 (2016).
[9] LeCun, Yann. ”The MNIST database of handwritten digits.” http://yann. lecun. com/exdb/mnist/ (1998).
[10] Lee, Stefan, et al. ”Why M heads are better than one: Training a diverse ensemble of deep networks.” arXiv preprint arXiv:1511.06314 (2015).
[11] Lu, Xiaofeng, et al. ”Privacy-Preserving Asynchronous Federated Learning Mechanism for Edge Network Computing.” IEEE Access 8 (2020): 48970-48981.
[12] McMahan, H. Brendan, et al. ”Communication-efficient learning of deep networks from decentralized data.” arXiv preprint arXiv:1602.05629 (2016)
[13] Ryffel, Theo, et al. ”A generic framework for privacy preserving deep learning.” arXiv preprint arXiv:1811.04017 (2018)
[14] Savazzi, Stefano, Monica Nicoli, and Vittorio Rampa. ”Federated learning with cooperating devices: A consensus approach for massive IoT networks.” IEEE Internet of Things Journal 7.5 (2020): 4641-4654.
[15] Simpson, Andrew JR. ”Instant learning: Parallel deep neural networks and convolutional bootstrapping.” arXiv preprint arXiv:1505.05972 (2015).
[16] Yadan, Omry, et al. ”Multi-gpu training of convnets.” arXiv preprint arXiv:1312.5853 (2013).
[17] Zhang, Wei, et al. ”Staleness-aware async-sgd for distributed deep learning.” arXiv preprint arXiv:1511.05950 (2015).
[18] Zhao, Yang, et al. ”Mobile edge computing, blockchain and reputation-based crowdsourcing iot federated learning: A secure, decentralized and privacy-preserving system.” arXiv preprint arXiv:1906.10893 (2019).
[19] https://ai.googleblog.com/2017/05/the-machine-intelligence-behind-gboard.html
[20] https://www.blog.google/products/maps/explore-around-town-google-maps/
[21] https://www.eff.org/deeplinks/2018/10/google-bug-more-about-cover-crime
[22] https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election
[23] https://www.uber.com/newsroom/2016-data-incident/