Putting Adaptive Federated Learning in a 2G Context

Emmanuel Azuh Mensah
ICTD University of Washington

ABSTRACT

With the proliferation of machine learning algorithms and their subsequent usage in every day lives, it is clear the enormous benefits of investing in machine learning. In the current global landscape however, wealthy nations are able to afford expensive computational resources for use in training and using machine learning models in various applications. Smaller countries are not only at a disadvantage because a small set of people are able to learn about machine learning but they also have extremely limited resources to support in training such models. This does not mean however that the need for these models in various applications is lacking in developing countries. This paper explores the performance implications of federated learning in an extreme environment setting - low resource edge devices (eg raspberry Pi) in poor networking environments (2G).

CCS CONCEPTS

• **Edge Machine Learning** → *Embedded systems*; • **Federated Learning**; • **Networks** → 2G;

KEYWORDS

federated learning, development computing, machine learning, poor networks

ACM Reference Format:
Emmanuel Azuh Mensah. 2021. Putting Adaptive Federated Learning in a 2G Context. In *Proceedings of Computer Communication and Networks (CSE 561 '21)*. UW CSE, Seattle, WA, USA, 6 pages. https://doi.org/10.000/0000

1 INTRODUCTION

The explosion of end user and IoT devices has triggered a huge interest in pushing compute from data centers closer to the network edge in an effort to reduce privacy breaches, save network congestion from massive data transfers, etc. This movement towards edge compute has no less seen increased machine learning algorithms deployed to the edge. One regime of machine learning on the edge which has recently gained a lot of traction is federated learning. Federated learning algorithms seek to coordinate multiple devices on the edge to learn a global model by pooling together resources without needing to share data with each other. Most of these advances however are located in richer countries, and as a result, the models don't take into account conditions prevalent in developing countries. We put federated learning in the context of developing countries in this paper to investigate how such algorithms perform and explore ways to make the models usable in the global south.

The rest of the paper is organized as follows: we provide some background for edge machine learning, then briefly describe the objective of the original paper we will be investigating including explaining concepts that are machine learning specific. Next, we make measurements to investigate the effects of algorithm parameters as well as network characteristics on the training performance. Then we investigate how a compression algorithm can reduce the training time of the algorithm under poor network conditions.

2 BACKGROUND

Edge machine learning has seen a huge explosion in interest in recent years due to its appeal to preserve user privacy as well as the promised efficiency of not needing to do all the training in large compute clusters but take advantage of compute power of combined many small devices. By 2025, there is estimated to be 80 billion connected devices generating about 160 zettabytes of data, 10 times as much data as was generated in 2016, forcing a huge need to move learning to the network edge [11].

Among the innovations to support edge machine learning are learning aware networking concepts such as using analog aggregation of edge updates over the air using broadband transmission to reduce communication latency [12]. Other works look at optimal selection of parameters such as batch size to efficiently train on edge devices while taking communication allocation into account [9]. [7] also tackles this problem by partially migrating model aggregation to edge servers from the cloud and formulates a joint computation and communication resource scheduling objective for devising a global cost minimization algorithm.

All these efforts go towards making the next wave of ML/AI applications possible. [5] proposes edge learning as a service to help push health care diagnostics and advice closer to users by users directly getting health information from devices in the comfort of their homes. Of the very few applications of edge computing in developing countries includes [1] which uses edge computing to monitor the health of electricity transformer stations in India but only as a static model (instead of one that constantly updates).

Many applications of ML in embedded systems can be translated into development context, especially since the applications already consider resource constrained devices. There are still assumptions about available resources that don't translate well to developing countries and these assumptions need to be investigated and new solutions proposed for them.

3 ADAPTIVE FEDERATED LEARNING

We briefly introduce the algorithm in [10] that we base our experiments[1] on in this chapter.

[1] https://github.com/emma-mens/adaptive-federated-learning

CSE 561 '21, March 2021, Seattle, Washington USA

ACM ISBN 000-0000-00-000/00/00.
https://doi.org/10.000/0000

On a high level, the workers in federated learning periodically receive the global weights from the aggregator along with the number of time steps till the next aggregation step, τ. Each worker computes a new weight update along with other parameters needed for computing τ as well as estimation of how much of predefined resource constraints are left. All these information are pushed to the aggregator at the next aggregation step. The worker algorithm is shown in figure 1.

$F(\mathbf{w})$	Global loss function
$F_i(\mathbf{w})$	Local loss function for node i
t	Iteration index
$\mathbf{w}_i(t)$	Local model parameter at node i in iteration t
$\mathbf{w}(t)$	Global model parameter in iteration t
$\mathbf{w}^{\mathrm{f}}$	Final model parameter obtained at the end of learning process
$\mathbf{w}^*$	True optimal model parameter that minimizes $F(\mathbf{w})$
η	Gradient descent step size
τ	Number of local update steps between two global aggregations
T	Total number of local update steps at each node
K	Total number of global aggregation steps, equal to T/τ
M (m)	Total number of resource types (the m-th type of resource)
R_m	Total budget of the m-th type of resource
c_m	Consumption of type-m resource in one local update step
b_m	Consumption of type-m resource in one global aggregation step
ρ	Lipschitz parameter of $F_i(\mathbf{w})$ $(\forall i)$ and $F(\mathbf{w})$
β	Smoothness parameter of $F_i(\mathbf{w})$ $(\forall i)$ and $F(\mathbf{w})$
δ	Gradient divergence
$h(\tau)$	Function defined in (11), gap between the model parameters obtained from distributed and centralized gradient descents
φ	Constant defined in Lemma 2, control parameter
$G(\tau)$	Function defined in (18), control objective
τ^*	Optimal τ obtained by minimizing $G(\tau)$

Initialize $t \leftarrow 0$;
repeat
  Receive $\mathbf{w}(t)$ and new τ^* from aggregator, set $\widetilde{\mathbf{w}}_i(t) \leftarrow \mathbf{w}(t)$;
  $t_0 \leftarrow t$; //Save iteration index of last transmission of $\mathbf{w}(t)$
  if $t > 0$ **then**
    Estimate $\hat{\rho}_i \leftarrow \|F_i(\mathbf{w}_i(t)) - F_i(\mathbf{w}(t))\| \,/\, \|\mathbf{w}_i(t) - \mathbf{w}(t)\|$;
    Estimate $\hat{\beta}_i \leftarrow \|\nabla F_i(\mathbf{w}_i(t)) - \nabla F_i(\mathbf{w}(t))\| \,/\, \|\mathbf{w}_i(t) - \mathbf{w}(t)\|$;
  for $\mu = 1, 2, ..., \tau^*$ **do**
    $t \leftarrow t + 1$; //Start of next iteration
    Perform local update and obtain $\mathbf{w}_i(t)$ according to (4);
    if $\mu < \tau^*$ **then**
      Set $\widetilde{\mathbf{w}}_i(t) \leftarrow \mathbf{w}_i(t)$;
  for $m = 1, 2, ..., M$ **do**
    Estimate type-m resource consumption $\hat{c}_{m,i}$ for one local update at node i;
  Send $\mathbf{w}_i(t)$, $\hat{c}_{m,i}$ $(\forall m)$ to aggregator;
  if $t_0 > 0$ **then**
    Send $\hat{\rho}_i$, $\hat{\beta}_i$, $F_i(\mathbf{w}(t_0))$, $\nabla F_i(\mathbf{w}(t_0))$ to aggregator;
until *STOP flag is received*;
Receive $\mathbf{w}(t)$ from aggregator;
Send $F_i(\mathbf{w}(t))$ to aggregator;

Figure 1: Federated Learning at Workers [10]

The aggregator receives information from the workers at the end of each aggregation round and computes the next value for τ, the new global weights and an estimation for ensuring the algorithm stays within resource budget as shown in figure 2. For more details on the federated learning algorithm, please refer to [10].

For this paper, we focus on the learning objective to train a digit recognition system for the MNIST [4] database. The algorithm used is the support vector machine (SVM) [8] to perform classification into the 10 digit classes based on the learning objective

$$\frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{2}max\{0; 1 - y_j \mathbf{w}^T \mathbf{x}_j\}^2$$

where $\mathbf{w}$ is the weight vector of dimension $784 = 28 \times 28$ representing the dimension of the images, y_j is the label for an image and x_j is the example image. The objective is optimized using stochastic gradient descent [2] which takes steps in the direction of the gradient towards a minimum.

Input: Resource budget R, control parameter φ, search range parameter γ, maximum τ value $\tau_{\max}$
Output: $\mathbf{w}^{\mathrm{f}}$
Initialize $\tau^* \leftarrow 1$, $t \leftarrow 0$, $s \leftarrow 0$; //s is a resource counter
Initialize $\mathbf{w}(0)$ as a constant or a random vector;
Initialize $\mathbf{w}^{\mathrm{f}} \leftarrow \mathbf{w}(0)$;
repeat
  Send $\mathbf{w}(t)$ and τ^* to all edge nodes, also send `STOP` if it is set;
  $t_0 \leftarrow t$; //Save iteration index of last transmission of $\mathbf{w}(t)$
  $t \leftarrow t + \tau^*$; //Next global aggregation is after τ iterations
  Receive $\mathbf{w}_i(t)$, $\hat{c}_i$ from each node i;
  Compute $\mathbf{w}(t)$ according to (5);
  if $t_0 > 0$ **then**
    Receive $\hat{\rho}_i$, $\hat{\beta}_i$, $F_i(\mathbf{w}(t_0))$, $\nabla F_i(\mathbf{w}(t_0))$ from each node i;
    Compute $F(\mathbf{w}(t_0))$ according to (2)
    if $F(\mathbf{w}(t_0)) < F(\mathbf{w}^{\mathrm{f}})$ **then**
      $\mathbf{w}^{\mathrm{f}} \leftarrow \mathbf{w}(t_0)$;
    if *STOP flag is set* **then**
      break; //Break out of the loop here if `STOP` is set
    Estimate $\hat{\rho} \leftarrow \frac{\sum_{i=1}^{N} D_i \hat{\rho}_i}{D}$;
    Estimate $\hat{\beta} \leftarrow \frac{\sum_{i=1}^{N} D_i \hat{\beta}_i}{D}$;
    Compute $\nabla F(\mathbf{w}(t_0)) \leftarrow \frac{\sum_{i=1}^{N} D_i \nabla F_i(\mathbf{w}(t_0))}{D}$, estimate $\hat{\delta}_i \leftarrow \|\nabla F_i(\mathbf{w}(t_0)) - \nabla F(\mathbf{w}(t_0))\|$ for each i, from which we estimate $\hat{\delta} \leftarrow \frac{\sum_{i=1}^{N} D_i \hat{\delta}_i}{D}$;
    Compute new value of τ^* according to (19) via linear search on integer values of τ within $[1, \tau_{\mathrm{m}}]$, where we set $\tau_{\mathrm{m}} \leftarrow \min\{\gamma\tau^*; \tau_{\max}\}$;
  for $m = 1, 2, ..., M$ **do**
    Estimate resource consumptions $\hat{c}_m$, $\hat{b}_m$, using $\hat{c}_{m,i}$ received from all nodes i and local measurements at the aggregator;
    $s_m \leftarrow s_m + \hat{c}_m \tau + \hat{b}_m$;
  if $\exists m$ *such that* $s_m + \hat{c}_m(\tau + 1) + 2\hat{b}_m \geq R_m$ **then**
    Decrease τ^* to the maximum possible value such that the estimated resource consumption for remaining iterations is within budget R_m for all m, set `STOP` flag;
Send $\mathbf{w}(t)$ to all edge nodes;
Receive $F_i(\mathbf{w}(t))$ from each node i;
Compute $F(\mathbf{w}(t))$ according to (2)
if $F(\mathbf{w}(t)) < F(\mathbf{w}^{\mathrm{f}})$ **then**
  $\mathbf{w}^{\mathrm{f}} \leftarrow \mathbf{w}(t)$;

Figure 2: Federated Learning at Aggregator [10]

4 EXPERIMENTAL SETUP

All experiments in this paper are ran on an Amazon Web Service EC2 t2.medium instance [2]. We ran one aggregator and five edge nodes as processes on the same instance. The instance was running an Amazon Linux operating system. For the modification of networking characteristics, we used the Linux Traffic Control tool [3]. Throughout our experiments, we assume the nodes are resource constrained and use that to guide decision making. In terms of the data distribution types described in [10], we used the uniformly distributed case for this paper.

5 MEASURING EFFECTS OF ALGORITHM PARAMETERS ON TRAINING PERFORMANCE

In this section, we explore the effects of algorithm parameters on training performance. The goal of these experiments was to trade some accuracy for looser restrictions on resource requirements (time, compute capability, etc). The experiments estimate how the following parameters affect the accuracy of the model as well as the training time required:

(1) Search range parameter (γ) with value 10 in [10]. This value is used to search for the optimal value of τ, the time between global aggregations around the current value of τ. So if $\gamma =$

[2] https://aws.amazon.com/ec2/instance-types/t2/
[3] https://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html

10, the new value of τ will be within 1 and $min\{\gamma * \tau_{old}, \tau_{max}\}$ (line 20 of Figure 2). We use the values $[3, 7, 10]$ for the experiments. The motivation for selecting values lower than 10 is to make the search space smaller in order to save compute resources.

(2) τ_{max} the maximum allowable time between successive global aggregations with value 100 in [10]. We use $[100, 125, 150]$ in order to allow for slower compute at the nodes if needed.
(3) The time budget for training the algorithm. Again, we use larger values than the value 15 used in [10] to allow for slower nodes due to minimal compute power at some edge nodes. We choose values $[15, 30, 60]$.

γ	τ_{max}	Time budget (s)	Accuracy (%)	Wall Time (s)
3	100	15	87.23	29.2996
3	100	30	87.48	48.3260
3	100	60	87.58	102.9359
3	125	15	87.26	27.036
3	125	30	87.52	47.2637
3	125	60	87.54	100.549
3	200	15	87.32	27.918
3	200	30	87.52	46.793
3	200	60	87.70	101.198
7	100	15	87.16	29.366
7	100	30	87.50	47.2803
7	100	60	87.62	95.306
7	125	15	87.27	27.0763
7	125	30	87.48	45.419
7	125	60	87.62	100.177
7	200	15	87.26	28.974
7	200	30	87.46	46.729
7	200	60	87.80	94.642
10	100	15	87.32	26.848
10	100	30	87.53	46.6732
10	100	60	87.61	103.939
10	125	15	87.28	27.716
10	125	30	87.51	49.122
10	125	60	87.50	98.886
10	200	15	87.31	28.183
10	200	30	87.53	45.492
10	200	60	87.51	99.672

Table 1: *Effect of algorithm parameters on training performance.*

From Table 1, we see that if no bound is set on the total time allowed for the algorithm to run, we can achieve similar accuracy values even with varying the selected algorithm parameters. Increasing γ slightly decreased the total time needed for training. Increasing the time budget as expected increases the total time till convergence of the algorithm. Varying τ_{max} had a less clear trend as it seemed to depend on the values of γ (at lower values, increasing τ_{max} tends to decrease training time but at higher values led to higher training times) and an even less clear trend when varying time budget.

For the remainder of our experiments, we use $\gamma = 3$, $\tau_{max} = 200$ and Time budget of 60 seconds following the logic presented at the beginning of this section.

6 MEASURING EFFECTS OF NETWORK CHARACTERISTICS ON TRAINING PERFORMANCE

Next, we look at how various networking constraints affect the convergence time of the algorithm if we want to keep the performance at a similar level to when we impose no network constraints.

To simulate network degradation, we impose any network change only on the central aggregator since workers don't communicate among themselves. The assumption is that if there is loss introduced at the ingress and egress of the aggregating server, both ways of the connection between the aggregator and a worker will experience the degradation so applying it only on the aggregating server should be sufficient.

The experimental guide for network characteristics[4] for 2G and 3G are as follows:

(1) Performance by bandwidth (2G:100kb/s, 200kb/s, 400kb/s; 3G:0.5Mb/s, 2Mb/s, 5Mb/s)
(2) Performance by loss (5%, 10%, 20%)
(3) Performance by latency (2G:300ms, 500ms, 1000ms; 3G:100ms, 200ms)

Bandwidth (Kb/s)	Loss (%)	Latency (ms)	Accuracy (%)	Wall Time (s)
-	-	300	0.877	297.359
-	-	500	0.8755	465.042
-	-	1000	0.8755	831.553
1000000	-	-	0.8759	97.781
1000	-	-	0.8752	254.055
100	-	-	0.8758	2022.105
-	5	-	0.8757	145.206
-	10	-	0.8753	179.623
-	20	-	0.8756	221.851

Table 2: *Effect of network characteristics on training performance.*

From table 2, we see that as expected, under tougher restrictions of bandwidth, loss and latency, it takes a longer time for convergence of the algorithm. Especially, if we use 100Kb/s it takes about 20 times longer compared to if the connection was 1Gb/s. For the rest of the paper, we focus on a method to improve convergence time under tighter bandwidth conditions since it had the most drastic impact on algorithm convergence time.

7 MEASURING EFFECTS OF ADAPTIVE COMPRESSION ON TRAINING TIME

Given the effects of low bandwidth on the convergence time of the federated learning algorithm, we consider compression as a way to reduce the time requirement. We make the observation that most of the weights are close to zero during training as shown in Figure 3. Our first approach was to compress by cutting the smallest absolute value weights closest to zero. This method resulted in worse training time *and* worse accuracy, leaving us in a worse position as shown in Figure 4.

Next we adapt the algorithm shown in Figures 5 and 6, taken from [6] to perform compression based on gradients in this paper. On a high level, each worker performs gradient compression by

[4] https://hpbn.co/mobile-networks/

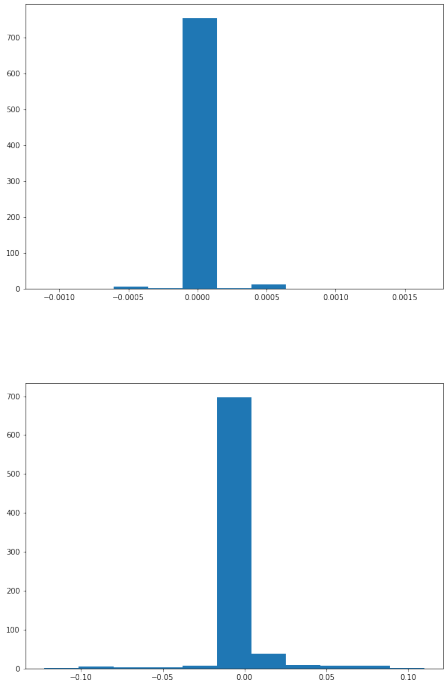

Figure 3: Histogram of weight distribution at the beginning of training (top) and at the end of training (bottom)

selecting the gradients corresponding to the top c largest gradient values. This compressed gradient matrix is sent to the parameter server.

The parameter server receives the compressed gradients from all the workers asynchronously, giving more emphasis to more recent gradients than older ones. For more details, refer to [6].

In this section, we adapt the algorithm above[5] into our federated learning setup. The original federated learning setup is synchronous whereas the Adaptive compression algorithm is asynchronous. We therefore made adjustments to only compress the weights but not take into account the staleness compensation in Adaptive compression.

Another important distinction between the federated learning setup we used in this project and that in [6] is that the clients don't perform any weight updates in AdaComp. They only compute gradients and compress by pruning small "gradients" to the parameter server. In our setup, the clients perform weight updates and the parameter server aggregates the weight updates. Our adaptation is to only send new weights that differ from the previous global model weights. That is, we compute

$$abs|\mathbf{w}_{global} - \mathbf{w}_{local}|$$

and compress by only sending weights corresponding to the top $c\%$ largest absolute value change in weight. The weights are sent along with their indices in the weight vector. At the parameter server, any missing index indicates a low change in value and so the previous

[5]https://github.com/Hardy-c/AdaComp

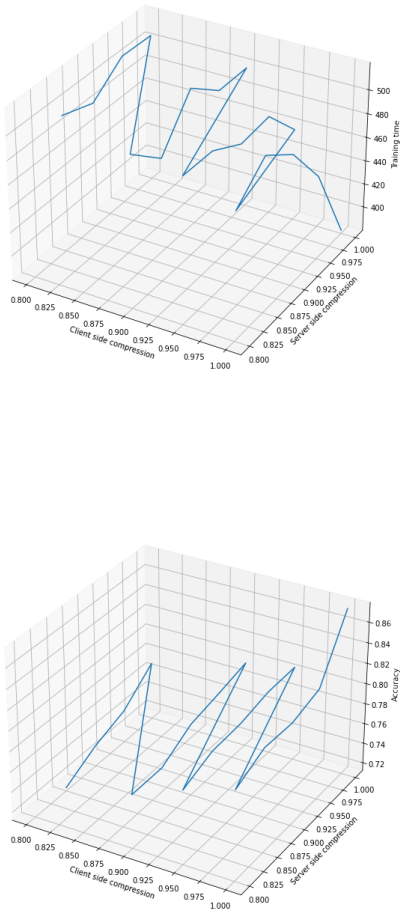

Figure 4: Weight compression vs. Training time (top) and Weight compression vs. Accuracy (bottom)

weight value in the global model is used. We found this method to be effective in actually reducing the training time by about half without reducing classification accuracy by much as seen in Figures 7 and 8.

From Figure 9, we see that ingress traffic to the parameter server doesn't drop by much indicating that even though we send about 0.05% of the original weight vector, we are sending small vectors more frequently, suggesting that the pruned weights end up causing the algorithm to take more global aggregations to converge but each aggregation is now much quicker than with no compression. Compressing too close to 0 also leads to a spike in ingress traffic suggesting the algorithm start to require too many global aggregations to learn so there is a sweet spot for low training time and low ingress traffic size.

procedure WORKER($PS, \mathcal{D}_w, I, b, c$)
  $i \leftarrow 0$
  while $i < I$ **do**
    $\Theta^{(w)}, i \leftarrow$ Pull_$\Theta(PS)$ ▷ get current $\Theta(i)$ from PS
    $\Delta\Theta^{(w)} \leftarrow$ SGD_STEP(D_w, b)
    $\Delta\tilde{\Theta}^{(w)} \leftarrow$ SELECT_GRAD($\Delta\Theta^{(w)}, c$)
    Push($\Delta\tilde{\Theta}^{(w)}, i$) ▷ update push to PS, with fetched i
  end while
end procedure
procedure SGD_STEP($\mathcal{D}_w, b$)
  Select a mini-batch B by sampling b examples from $\mathcal{D}_w$ and compute $\Delta\Theta^{(w)}$ with back-propagation method.
  return $\Delta\Theta^{(w)}$
end procedure
procedure SELECT_GRAD($\Delta\Theta^{(w)}, c$)
  Select $\Delta\tilde{\Theta}^{(w)} \subset \Delta\Theta^{(w)}$ by keeping the $(100 \times c)\%$ largest parameters, in absolute value, of each matrix V_l and vector β_l.
  return $\Delta\tilde{\Theta}^{(w)}$
end procedure

Figure 5: AdaComp at Wokers [6]

procedure PS(α, I)
  Initialize $\Theta(0)$ with random values.
  for $i \leftarrow 0, I$ **do**
    Get($\Delta\tilde{\Theta}^{(w)}, j$) ▷ wait for a push
    $\Delta\Theta(i) \leftarrow \Delta\tilde{\Theta}^{(w)}$
    for all $\Delta\Theta_k(i) \in \Delta\Theta(i)$ **do**
      $\sigma_k \leftarrow \sum_{u=j}^{i-1} \mathbb{1}_{\{\Delta\Theta_k(u)\neq 0\}}$
      if $\sigma_k = 0$ **then** $\alpha_k \leftarrow \alpha$ **else** $\alpha_k \leftarrow \alpha/\sigma_k$
      $\Theta_k(i+1) \leftarrow \Theta_k(i) - \alpha_k \Delta\Theta_k(i)$
    end for
  end for
end procedure

Figure 6: AdaComp at Parameter Server [6]

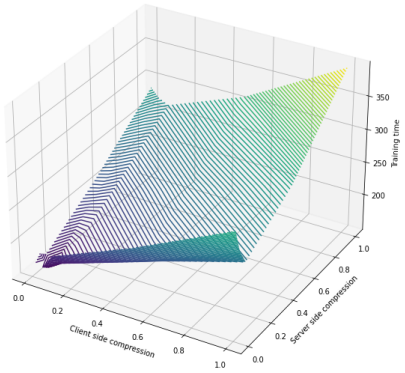

Figure 7: Gradient based compression vs Training time

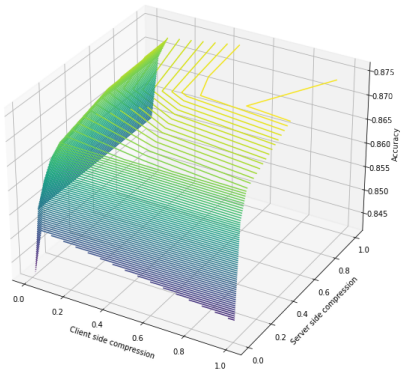

Figure 8: Gradient based compression vs Accuracy

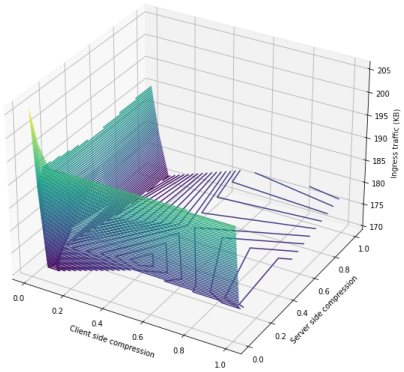

Figure 9: Gradient based compression vs Ingress Traffic

8 CONCLUSION AND FUTURE DIRECTIONS

With the initial findings of how poor networking conditions affect federated learning algorithms, we believe we are in a better position to think of ways to contribute to this line of work, especially in the developing countries context by devising machine learning aware networking algorithms that work well in rugged terrains (low resource countries and extraterrestrial applications such as in Mars exploration alike). Below are a few future directions.

(1) Including optimized compression in microTVM [3]. In this project, we didn't impose a strong constraint on the memory capability available to each client server. In actual small device edge learning, even the compression algorithm may be expensive and a more optimized version (eg operating in sparse space) may be required. We would like to include such capabilities in the microTVM architecture.

(2) Overcoming high latency high loss. We focused on low bandwidth in this project by compressing the data communicated. A possibly harder constrain to solve is dealing with high loss when the network exhibits high latency and has low bandwidth. We would like to explore low memory caching mechanisms in such cases.
(3) Understand better the interaction amongst all three factors - bandwidth, loss and latency - on algorithm performance. This will involve a better understanding of the linux traffic control tool. We found the documentation on the traffic control tool to be difficult to follow. Going forward, we will work with people who have used the tool to get better control on network characteristics to use in the experiments.
(4) Using actual multiple devices instead of a simulated environment of multiple processes on the same device representing the parameter server and clients.
(5) Verifying that our observations generalize to all the different data distribution cases.
(6) Studying the case where we use Convolutional Neural Networks which are a lot more memory intensive machine learning models.

ACKNOWLEDGMENTS

Many thanks to Ratul for his advice on tools to use, Kurtis Heimerl for his support with AWS computing resources and Esther Jang for lending her Raspberry Pi which we will utilize in future works.

REFERENCES

[1] Imtiyaz Ahmad, Yaduvir Singh, and Jameel Ahamad. 2020. Machine Learning Based Transformer Health Monitoring Using IoT Edge Computing. In *2020 5th International Conference on Computing, Communication and Security (ICCCS)*. IEEE, 1–5.
[2] Léon Bottou. 2012. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*. Springer, 421–436.
[3] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 578–594. https://www.usenix.org/conference/osdi18/presentation/chen
[4] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.
[5] Li Gaolei, Guangquan Xu, Kunmar Sangaiah, Jun Wu, and Jianhua Li. 2019. EdgeLaaS: Edge Learning as a Service for Knowledge-Centric Connected Healthcare. *IEEE Network* 33 (07 2019). https://doi.org/10.1109/MNET.001.1900019
[6] Corentin Hardy, Erwan Le Merrer, and Bruno Sericola. 2017. Distributed deep learning on edge-devices: feasibility via adaptive compression. In *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*. IEEE, 1–8.
[7] Siqi Luo, Xu Chen, Qiong Wu, Zhi Zhou, and Shuai Yu. 2020. Hfel: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning. *IEEE Transactions on Wireless Communications* 19, 10 (2020), 6535–6548.
[8] William S Noble. 2006. What is a support vector machine? *Nature biotechnology* 24, 12 (2006), 1565–1567.
[9] Jinke Ren, Guanding Yu, and Guangyao Ding. 2020. Accelerating DNN training in wireless federated edge learning systems. *IEEE Journal on Selected Areas in Communications* 39, 1 (2020), 219–232.
[10] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. 2019. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications* 37, 6 (2019), 1205–1221.
[11] Guangxu Zhu, Dongzhu Liu, Yuqing Du, Changsheng You, Jun Zhang, and Kaibin Huang. 2020. Toward an intelligent edge: Wireless communication meets machine learning. *IEEE communications magazine* 58, 1 (2020), 19–25.
[12] G. Zhu, Y. Wang, and K. Huang. 2020. Broadband Analog Aggregation for Low-Latency Federated Edge Learning. *IEEE Transactions on Wireless Communications* 19, 1 (2020), 491–506. https://doi.org/10.1109/TWC.2019.2946245