Decentralized learning works: An empirical comparison of gossip learning and federated learning

MO809 - Tópicos em Computação Distribuída

Luiz Fernando Rodrigues da Fonseca (RA 156475)

Index

Introduction

Introduction

István Hegedűs, Gábor Danner, Márk Jelasity, Decentralized learning works: An empirical comparison of gossip learning and federated learning, Journal of Parallel and Distributed Computing, Volume 148, 2021, Pages 109-124, ISSN 0743-7315, https://doi.org/10.1016/j.jpdc.2020.10.006.

Journal of Parallel and Distributed Computing 148 (2021) 109–124

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Decentralized learning works: An empirical comparison of gossip learning and federated learning☆

István Hegedűs [a], Gábor Danner [a], Márk Jelasity [a,b,*]

[a] University of Szeged, Szeged, Hungary
[b] MTA SZTE Research Group on Artificial Intelligence, Szeged, Hungary

ARTICLE INFO

Article history:
Received 15 October 2019
Received in revised form 8 October 2020
Accepted 18 October 2020
Available online 4 November 2020

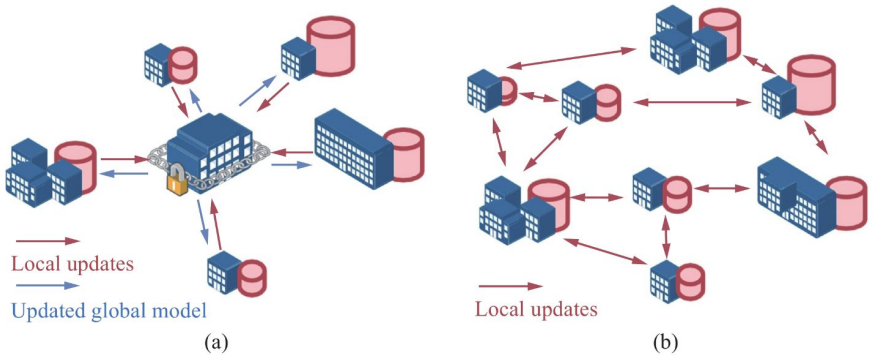
Keywords:
Federated learning
Gossip learning
Decentralized machine learning

ABSTRACT

Machine learning over distributed data stored by many clients has important applications in use cases where data privacy is a key concern or central data storage is not an option. Recently, federated learning was proposed to solve this problem. The assumption is that the data itself is not collected centrally. In a master–worker architecture, the workers perform machine learning over their own data and the master merely aggregates the resulting models without seeing any raw data, not unlike the parameter server approach. Gossip learning is a decentralized alternative to federated learning that does not require an aggregation server or indeed any central component. The natural hypothesis is that gossip learning is strictly less efficient than federated learning due to relying on a more basic infrastructure: only message passing and no cloud resources. In this empirical study, we examine this hypothesis and we present a systematic comparison of the two approaches. The experimental scenarios include a real churn trace collected over mobile phones, continuous and bursty communication patterns, different network sizes and different distributions of the training data over the devices. We also evaluate a number of additional techniques including a compression technique based on sampling, and token account based flow control for gossip learning. We examine the aggregated cost of machine learning in both approaches. Surprisingly, the best gossip variants perform comparably to the best federated learning variants overall, so they offer a fully decentralized alternative to federated learning.

Introduction

- Interest in data privacy led to the proposal of Federated Learning (FL)
- FL is based on a centralized server that holds the global model and aggregates the trained parameters of the nodes
- Gossip Learning (GL) addresses the same challenges but in a decentralized way, nodes exchange and aggregate models directly



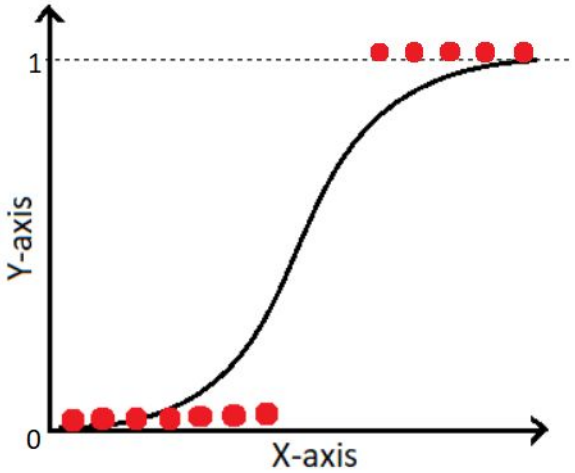
Introduction

- Features of GL:
  - Peer to Peer
  - Data is distributed horizontally
  - No infrastructure required (no single point of failure)
  - Cheaper scalability and better robustness
- How do FL and GL compare in terms of convergence time and model quality?
- Proposals to improve GL and experiments to validate them

Machine Learning Model

Machine Learning Model

- Classification problem with Logistic Regression
- Regularization and Logistic Loss
- Training with Stochastic Gradient Descent (SGD)



$$J(w, b) = \frac{\lambda}{2}\|w\|^2 - \frac{1}{n}\sum_{i=1}^{n}\left[y_i \ln f_{(w,b)}(x_i) + (1 - y_i) \times \ln(1 - f_{(w,b)}(x_i))\right],$$

where $y_i \in \{0, 1\}$ and

$$f_{(w,b)}(x_i) = P(y_i = 1 | x_i, w, b) = \frac{1}{1 + e^{(w^T x + b)}}.$$

Gossip Learning

Gossip Learning

- Initialize the model and age ***t***
- Periodically send the model to a peer:
  - Wait some time (rounds are not synchronized)
  - Select a neighbor peer
  - Send a subset of the parameters to the other peer
- When a node receives a model:
  - Merge with the local model
  - Local update (training)

Algorithm 1 Gossip Learning

1: $(t_k, w_k, b_k) \leftarrow (\mathbf{0}, \mathbf{0}, 0)$
2: **loop**
3: wait(Δ_g)
4: $p \leftarrow$ selectPeer()
5: send sample(t_k, w_k, b_k) to p
6: **end loop**
7:
8: **procedure** ONRECEIVEMODEL(t_r, w_r, b_r)
9: $(t_k, w_k, b_k) \leftarrow$ merge$((t_k, w_k, b_k), (t_r, w_r, b_r))$
10: $(t_k, w_k, b_k) \leftarrow$ update$((t_k, w_k, b_k), D_k)$
11: **end procedure**

Gossip Learning

- It is assumed a static, connected and random overlay network to simplify the ***selectPeer*** (random)
- Different implementations of ***sample***, ***merge*** and ***update***
- 3 optimizations applied: sampling, model partitioning and token account algorithms

Algorithm 1 Gossip Learning

1: $(t_k, w_k, b_k) \leftarrow (\mathbf{0}, \mathbf{0}, 0)$
2: **loop**
3: wait(Δ_g)
4: $p \leftarrow$ selectPeer()
5: send sample(t_k, w_k, b_k) to p
6: **end loop**
7:
8: **procedure** ONRECEIVEMODEL(t_r, w_r, b_r)
9: $(t_k, w_k, b_k) \leftarrow$ merge($(t_k, w_k, b_k), (t_r, w_r, b_r)$)
10: $(t_k, w_k, b_k) \leftarrow$ update($(t_k, w_k, b_k), D_k$)
11: **end procedure**

Gossip Learning Improvements

Sampling and Model Partitioning

- Only ***w*** is partitioned (***b*** is always included on the sample)
- 2 implementations of ***sample***:
  - ***sampleRandom(t, w, b, s)***:
    - Returns a uniform random subset of the parameters
    - ***s*** is the percentage of parameters to be sampled from ***w***
  - ***samplePartition(t, w, b, j)***:
    - Includes the model partitioning
    - ***S*** is the number of partitions
    - ***j*** is the partition number (***j = i mod S***)

S = 3, w =

i = 0 j = 0	i = 1 j = 1	i = 2 j = 2	i = 3 j = 0	i = 4 j = 1

Sampling and Model Partitioning

- Weighted average of parameter vectors in ***merge***
- Merge only the partition received
- Each partition has a different age parameter ***t[j]***
- ***t*** is updated with the max of the local and the received values

Algorithm 2 Partitioned Model Merge

S : the number of partitions
procedure MERGE($(t, w, b), (t_r, w_r, b_r)$)
    $j \leftarrow$ index of received partition ▷ $j = i \mod S$, for any coordinate i within the sample
    for coordinate i is included in sample **do**
        $w[i] \leftarrow (t[j] \cdot w[i] + t_r[j] \cdot w_r[i])/(t[j] + t_r[j])$
    end for
    $b \leftarrow (t[S] \cdot b + t_r[S] \cdot b_r)/(t[S] + t_r[S])$
    $t \leftarrow \max(t, t_r)$ ▷ element-wise maximum, where t_r is defined
    return (t, w, b)
end procedure

Sampling and Model Partitioning

- SGD for the ***update***
- The dataset is split into batches ***B***
- All partitions have their own dynamic learning rate that is determined by the age of the partition

Algorithm 3 Partitioned Model Update Rule

S : the number of partitions
d : the dimension of w
procedure UPDATE($(t, w, b), D$)
  for all batch $B \subseteq D$ **do** ▷ D is split into batches
    $t \leftarrow t + |B| \cdot \mathbf{1}$ ▷ increase all ages by $|B|$
    for $i \in \{1, ..., d\}$ **do**
      $$h[i] \leftarrow -\frac{\eta}{t[i \mod S]} \sum_{(x,y)\in B} \left(\frac{\partial \ell(f_{w,b}(x), y)}{\partial w[i]}(w[i]) + \lambda w[i]\right)$$
    end for
    $$g \leftarrow -\frac{\eta}{t[S]} \sum_{(x,y)\in B} \left(\frac{\partial \ell(f_{w,b}(x), y)}{\partial b}(b) + \lambda b\right)$$
    $w \leftarrow w + h$
    $b \leftarrow b + g$
  end for
  return (t, w, b)
end procedure

Token Gossip Learning

- Token account algorithm allows the formation of chains of messages that travel fast in the network ("hot potato")
- Flow control mechanism with tokens that allow the "hot potatoes" to avoid waiting the next cycle at each node
- Proactive and reactive messages are sent
- ***samplePartition*** is used, ***sampleRandom*** prevents the formation of the "hot potato"

Token Gossip Learning

- Each partition ***j*** at node ***k*** has its own token account ***ak[j]***
- Tokens used to send the model to peers
- The nodes proactively might send a model to a peer, otherwise they win a token
- Or when they receive a model, they might reactively send the model to one or more peers

Algorithm 4 Partitioned Token Gossip Learning

```
1:  (t_k, w_k, b_k) ← (0, 0, 0)
2:  a_k ← 0
3:  loop
4:      wait(Δ_g)
5:      j ← selectPart()                          ▷ select a random partition
6:      do with probability proactive(a_k[j])
7:          p ← selectPeer()
8:          send samplePartition(t_k, w_k, b_k, j) to p
9:      else
10:         a_k[j] ← a_k[j] + 1    ▷ we did not spend the token so it accumulates
11:     end do
12: end loop
13:
14: procedure ONRECEIVEMODEL(t_r, w_r, b_r, j)
15:     (t_k, w_k, b_k) ← merge((t_k, w_k, b_k), (t_r, w_r, b_r))
16:     (t_k, w_k, b_k) ← update((t_k, w_k, b_k), D_k)
17:     x ← randRound(reactive(a_k[j]))
18:     a_k[j] ← a_k[j] − x                       ▷ we spend x tokens
19:     for i ← 1 to x do
20:         p ← selectPeer()
21:         send samplePartition(t_k, w_k, b_k, j) to p
22:     end for
23: end procedure
```

Token Gossip Learning

- ***proactive***: probability of sending a proactive message
- ***reactive***: number of reactive messages
- ***C*** is the maximum number of tokens and ***A*** is the motivation level for saving tokens

$$\text{PROACTIVE}(a) = \begin{cases} 0 & \text{if } a < A - 1 \\ \dfrac{a - A + 1}{C - A + 1} & \text{if } a \in [A - 1, C] \end{cases}$$

and $\text{REACTIVE}(a) = a/A$, with parameters $A = 10$ and $C = 20$.

Algorithm 4 Partitioned Token Gossip Learning

```
1: (t_k, w_k, b_k) ← (0, 0, 0)
2: a_k ← 0
3: loop
4:     wait(Δ_g)
5:     j ← selectPart()                    ▷ select a random partition
6:     do with probability proactive(a_k[j])
7:         p ← selectPeer()
8:         send samplePartition(t_k, w_k, b_k, j) to p
9:     else
10:        a_k[j] ← a_k[j] + 1    ▷ we did not spend the token so it accumulates
11:    end do
12: end loop
13:
14: procedure ONRECEIVEMODEL(t_r, w_r, b_r, j)
15:    (t_k, w_k, b_k) ← merge((t_k, w_k, b_k), (t_r, w_r, b_r))
16:    (t_k, w_k, b_k) ← update((t_k, w_k, b_k), D_k)
17:    x ← randRound(reactive(a_k[j]))
18:    a_k[j] ← a_k[j] − x                 ▷ we spend x tokens
19:    for i ← 1 to x do
20:        p ← selectPeer()
21:        send samplePartition(t_k, w_k, b_k, j) to p
22:    end for
23: end procedure
```

Token Gossip Learning

- With each partition having a separate account, each partition will perform random walks independently, using its own communication budget
- ***selectPart*** and ***selectPeer*** implemented with sampling without replacement (lower variance)

Algorithm 4 Partitioned Token Gossip Learning

```
1:  (t_k, w_k, b_k) ← (0, 0, 0)
2:  a_k ← 0
3:  loop
4:      wait(Δ_g)
5:      j ← selectPart()                          ▷ select a random partition
6:      do with probability proactive(a_k[j])
7:          p ← selectPeer()
8:          send samplePartition(t_k, w_k, b_k, j) to p
9:      else
10:         a_k[j] ← a_k[j] + 1    ▷ we did not spend the token so it accumulates
11:     end do
12: end loop
13:
14: procedure ONRECEIVEMODEL(t_r, w_r, b_r, j)
15:     (t_k, w_k, b_k) ← merge((t_k, w_k, b_k), (t_r, w_r, b_r))
16:     (t_k, w_k, b_k) ← update((t_k, w_k, b_k), D_k)
17:     x ← randRound(reactive(a_k[j]))
18:     a_k[j] ← a_k[j] − x                       ▷ we spend x tokens
19:     for i ← 1 to x do
20:         p ← selectPeer()
21:         send samplePartition(t_k, w_k, b_k, j) to p
22:     end for
23: end procedure
```

Federated Learning

Federated Learning

- Regular FL with small modifications
- Communication is compressed through sampling the parameters (***sup <= sdown***)
- Use of ***sampleRandom***, ***update*** is the same as GL with ***S = 1***

Algorithm 5 Federated Learning Master

$(t, w, b) \leftarrow$ init()
loop
  for every node k **in parallel do** ▷ non-blocking (in separate threads)
    send sample(t, w, b, s_{down}) to k
    receive (n_k, h_k, g_k) from k ▷ n_k: #examples at k; h_k: sampled model gradient; g_k: bias gradient
  end for
  wait(Δ_f) ▷ the round length
  $n \leftarrow \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} n_k$ ▷ $\mathcal{K}$: nodes that returned a model in this round
  $t \leftarrow t + n$
  $h \leftarrow$aggregate$(\{h_k : k \in \mathcal{K}\})$
  $w \leftarrow w + h$
  $g \leftarrow \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} g_k$
  $b \leftarrow b + g$
end loop

Algorithm 6 Federated Learning Worker

$(t_k, w_k, b_k) \leftarrow$ init() ▷ the local model at the worker

procedure ONRECEIVEMODEL(t, w, b) ▷ w: sampled model
  $t_k \leftarrow t$
  for $w[i] \in w$ **do** ▷ coordinate i is defined in w
    $w_k[i] \leftarrow w[i]$
  end for
  $b_k \leftarrow b$
  $(t_k, w_k, b_k) \leftarrow$ update$((t_k, w_k, b_k), D_k)$ ▷ D_k: the local database of examples
  $(n, h, g) \leftarrow (t_k - t, w_k - w, b_k - b)$ ▷ n: the number of local examples, h: the gradient update
  send sample(n, h, g, s_{up}) to master
end procedure

Federated Learning

- For the aggregation we have sampling
- ***aggregate*** is simpler and averages each received gradient
- ***aggregateImproved*** takes the average of only those coordinates that are included in the sample
- To get an unbiased estimate, divide by the probability that there is at least one gradient in which the given coordinate is included (correction of the learning rate)

Algorithm 7 Variants of the aggregate function

procedure AGGREGATE(H)
  $h' \leftarrow \mathbf{0}$
  for $i \in \{1, \ldots, d\}$ **do**
    $h'[i] \leftarrow \frac{1}{s|H|} \sum_{h \in H : h[i] \in h} h[i]$ ▷ $s \in (0, 1]$: sampling rate used to create H
  end for
  return h'
end procedure

procedure AGGREGATEIMPROVED(H)
  $h' \leftarrow \mathbf{0}$
  for $i \in \{1, \ldots, d\}$ **do**
    $H_i \leftarrow \{h : h \in H \wedge h[i] \in h\}$
    $h'[i] \leftarrow \frac{1}{|H_i|(1-(1-s)^{|H|})} \sum_{h \in H_i} h[i]$ ▷ skipped if $|H_i| = 0$
  end for
  return h'
end procedure

Experimental Setup

Experimental Setup

- PeerSim simulator
- Performance metric is the proportion of misclassified examples in the test set:
  - GL: Average loss over the online nodes
  - FL: Central model at the master
- The metric is a function of the total number of bits communicated (the full model is the unit of transferred information)
- Multiple experiments with variations

Experimental Setup - Data

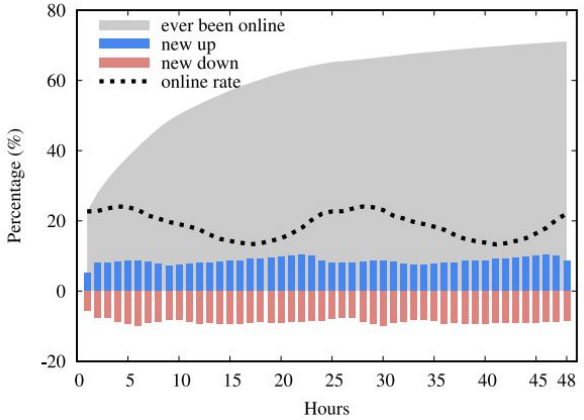
- Datasets:
  - Spambase: decide whether an email is spam or not (features word or character frequencies)
  - Pendigits: **4 x 4** downsampled images of digits from 0 to 9
  - HAR (Human Activity Recognition): human activities (walking, sitting, staying, etc) were monitored by smartphone sensors (accelerometer, gyroscope, etc)
- 3 data distribution schemes:
  - 100 nodes in the network: data evenly distributed
  - Network size equals dataset size: each node has one example
  - Hybrid: each data example was assigned to multiple nodes
- Label distribution: random uniform assignment or single class assignment

Experimental Setup - Network

- Fixed random k-out overlay network, with ***k = 20*** neighbors
- Types of experiments:
  - Always online (no failure) vs churn scenarios
  - Continuous (slow) vs bursty (fast) transfers
- Assumptions:
  - Nodes can detect easily if their neighbors are online
  - FL server has unlimited bandwidth
  - Nodes have the same download and upload bandwidth (lower cap than usual)
- Model transfer time in the churn scenario (bigger than real values):
  - 86.4s long transfer time (more transfers will fail)
  - 8.64s short transfer time

Experimental Setup - Phone Traces

- Traces collected by the app STUNer (1191 users)
- 2-day segments:
  - First day to achieve a token distribution that reflects a real application
  - Second day for ML training (FL/GL)
- Device is available if connected to a charger, connected more than 1 minute, and bandwidth higher than 1Mb/s



Experimental Setup - Hyperparameters

- Training round times:
  - Continuous transfer:
    - GL: Transfer time of the model
    - FL: Round-trip time (sum of download and upload times)
  - Bursty transfer: ***p*** percentage of time of the continuous transfer
- Sampling ***s*** in {1, 0.5, 0.25, 0.1}, ***s = 1/S***
- Grid search for the learning rate and regularization parameters

Table 2
Hyperparameters.

	Spambase	Pendigits	HAR
Parameter η	10^3	10^4	10^2
Parameter λ	10^{-3}	10^{-4}	10^{-2}

Experimental Results

Design Choice 1

- Comparison of ***aggregate*** and ***aggregateImproved*** in FL no failure continuous transfer
- Slight advantage of the improved version

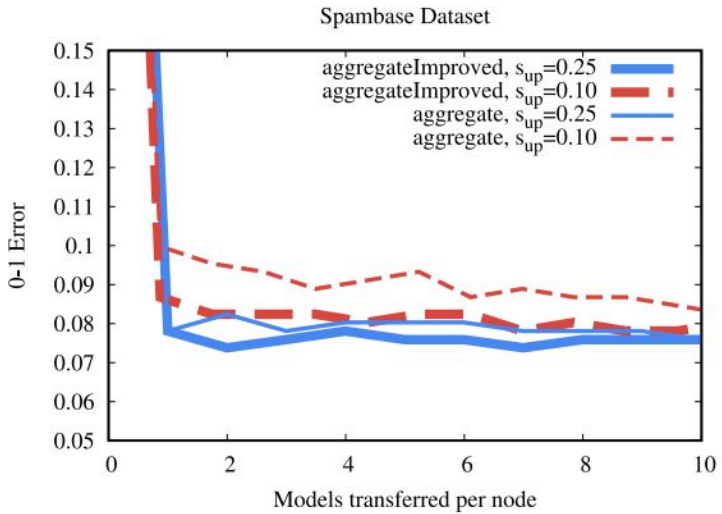


Fig. 4. Federated learning, 100 nodes, long transfer time, no failures, different aggregation algorithms and upstream subsampling probabilities and with $s_{down} = 1$.

Design Choice 2

- Comparison of partitioned and not partitioned GL
- Partitioned implementation is better, because each partition has an age and they are updated separately
- In the not partitioned GL, the model age is updated as a whole, but only a random subset of parameters are updated

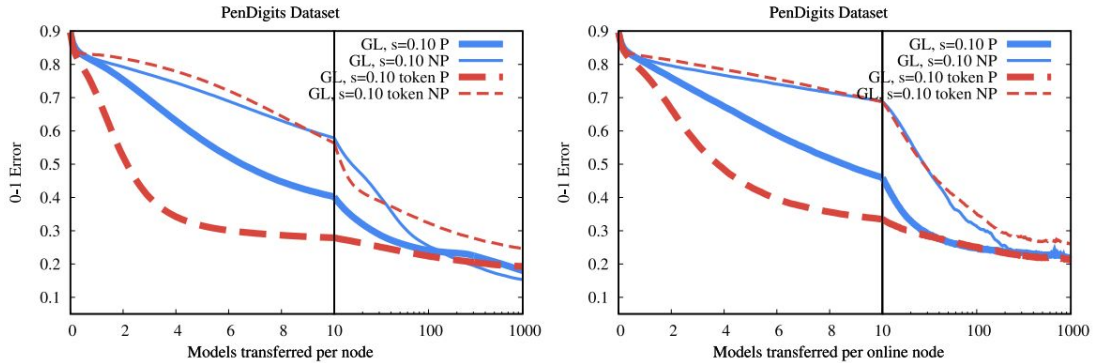


Fig. 5. Gossip learning with one node for each example, bursty transfer, subsampling probability $s = 0.1$, no-failure (left) and smartphone trace with long transfer time (right). Variants with and without model partitioning, indicated as P and NP, respectively.

Design Choice 3

- Subsampling comparison in FL (master to node and vice versa)
- In both directions is better (design choice), but the nodes don't receive the full model (this might not be a problem for some applications)

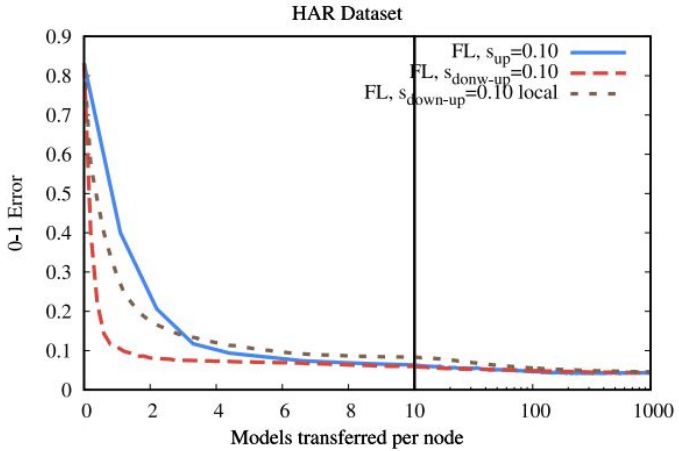


Fig. 6. Federated learning with 100 nodes, no-failure scenario, with different subsampling strategies and $s = 0.1$. The "local" plot indicates the average of the models that the clients store, otherwise the master's model is evaluated.

Continuous Transfer - Churn Free

- In the 100 nodes scenario (left), GL is competitive with FL when there is high compression rates
- In the 1 example per node scenario (right), FL is better due to the little local information on the nodes

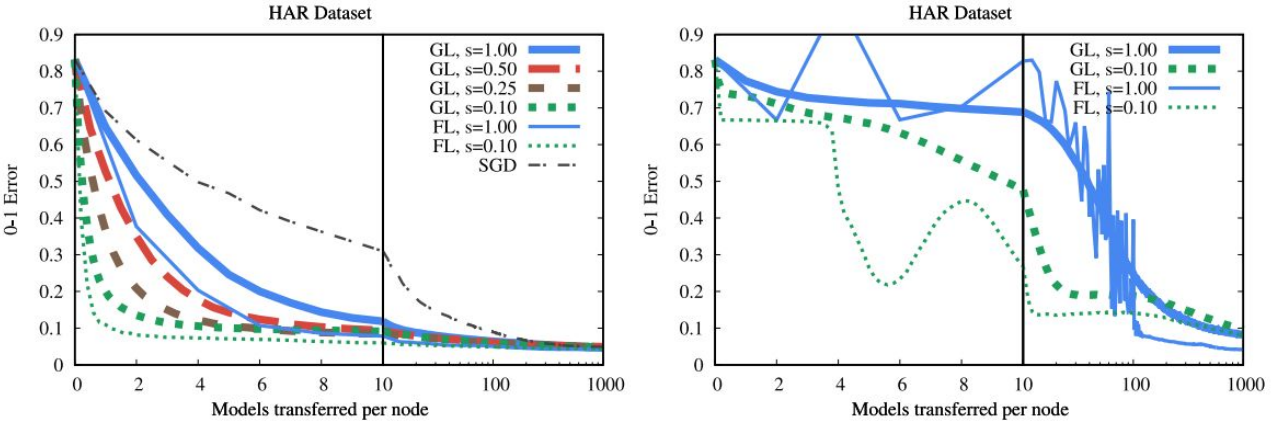


Fig. 7. Federated learning and gossip learning with 100 nodes (left) and with one node for each sample (right), no-failure scenario, with different subsampling probabilities. Stochastic Gradient Descent (SGD) is implemented by gossip learning with no merging (received model replaces current model).

Continuous Transfer - Churn

- Long or short transfer times causes almost no difference
- Churn only causes a minor increase in the variance of the error, but it's still a stable convergence (model partitioning helps)

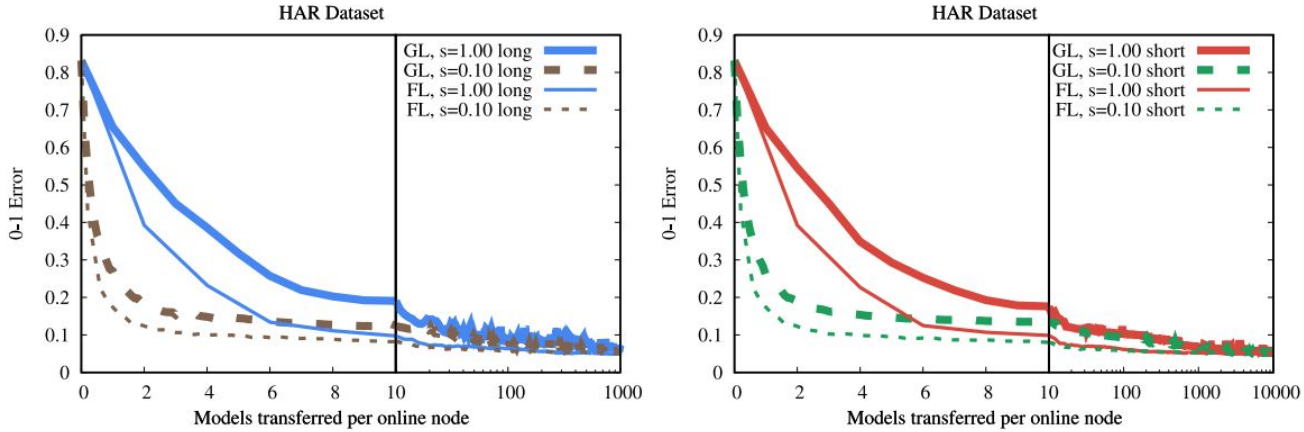


Fig. 8. Federated learning and gossip learning over the smartphone trace with long (left) and short (right) transfer time, in the 100 node scenario.

Continuous Transfer - Single Class

- The advantage of FL here is more apparent also due to little local information, but GL is also good on the long run

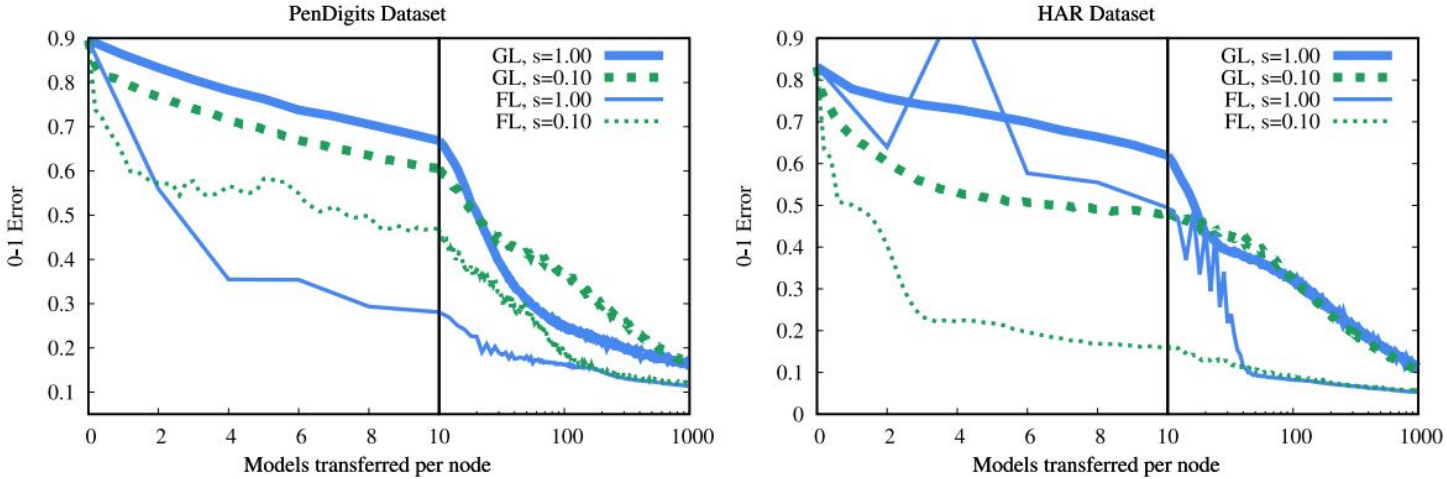


Fig. 9. Federated learning and gossip learning with 100 nodes, no-failure scenario, with single class assignment.

Bursty Transfer

- Similar results to the continuous transfer
- Token GL converges faster than regular GL
- GL is competitive with FL, except on the extreme scenarios

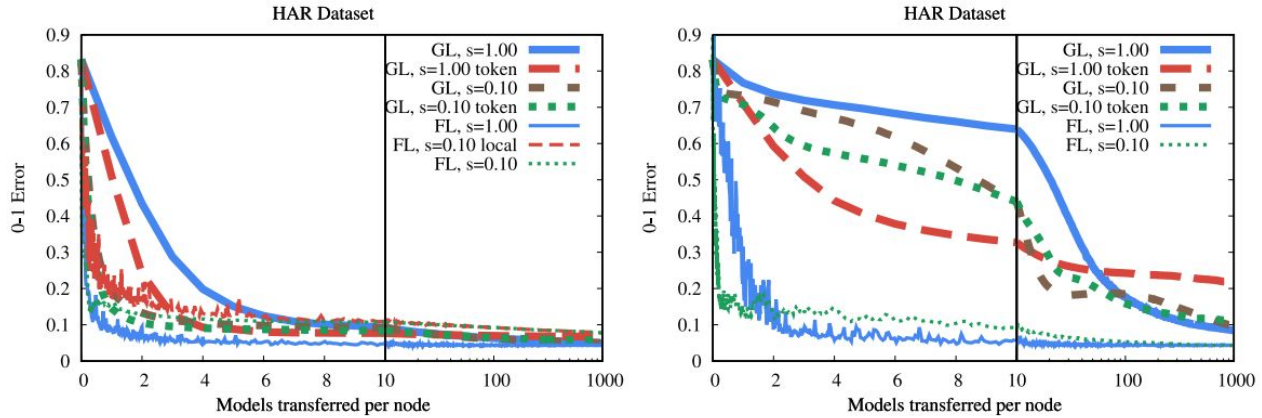


Fig. 10. Federated learning and gossip learning with 100 nodes (left) and with one node for each sample (right), no-failure scenario, in the bursty transfer scenario.

Conclusion

Conclusion

- Gossip Learning is a decentralized way of training Machine Learning models without sharing private data and without a central server, like Federated Learning
- Some modifications to the algorithms were proposed enhancing the performance of GL, like sampling, model partitioning and token account algorithms
- Multiple experiments were conducted to compare GL and FL (always online vs churn, random assignment vs single class assignment)
- GL is competitive with FL in some cases, except in the single class assignment scenarios
- Future work: simulations of realistic networks, heterogeneous nodes, data distribution schemes, network topologies, complex models

References

- István Hegedűs, Gábor Danner, Márk Jelasity, Decentralized learning works: An empirical comparison of gossip learning and federated learning, Journal of Parallel and Distributed Computing, Volume 148, 2021, Pages 109-124, ISSN 0743-7315, https://doi.org/10.1016/j.jpdc.2020.10.006.
- Giuseppi, A., Manfredi, S. & Pietrabissa, A. A Weighted Average Consensus Approach for Decentralized Federated Learning. *Mach. Intell. Res.* 19, 319–330 (2022). https://doi.org/10.1007/s11633-022-1338-z.
- A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," 2009 IEEE Ninth International Conference on Peer-to-Peer Computing, 2009, pp. 99-100, doi: 10.1109/P2P.2009.5284506.
- G. Danner and M. Jelasity, "Token Account Algorithms: The Best of the Proactive and Reactive Worlds," 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), 2018, pp. 885-895, doi: 10.1109/ICDCS.2018.00090.