CS460 Midway Presentation

On Federated Learning and Possible Improvements

Saswat Das[1] Suraj Patel[2]

[1]School of Mathematical Sciences
NISER, HBNI
[2]School of Physical Sciences
NISER, HBNI

Introduction : Work So Far

A Recapitulation:
Federated Learning (FL) is the practice of, in contrast to central learning with all the data in one place, letting users train local models on their devices and send the results of the training (weights/gradients) to the server for *aggregation*, without their raw data ever leaving their devices.

Introduced by Brendan McMahan et al. (*Communication-Efficient Learning of Deep Networks from Decentralized Data*)[1] in 2017.

Our aim: To explore various FL techniques, look for the best performing ones (w.r.t. communication rounds, accuracy, privacy, communication overhead etc.)

Promised Midway Targets:

1. Successfully implement a FL model, and come up with some preliminary results.
2. Suggest some tweaks and improvements to test out.
3. If needed, present pertinent concepts from relevant literature.

Literature Review:

McMahan et al[1]

- Introduces federated learning;
- Addresses issues such as unbalanced volume of datapoints across all clients, limited communication capabilities of clients (via multiple rounds of communication), and the mass distributed nature of the clients, and the non-IID nature of data as an individual's data is specific;
- Introduces `FedAvg` ($w_{t+1} \leftarrow \sum_{i=1}^{K} \frac{n_k}{n} w_{t+1}^k$);
- Talks about controlling parallelism of local computation and increased local computation by varying batch size for gradient descent (decreasing it increases amount/precision of computation, no. of rounds, and no. of clients queried per round (affects parallelism).

Wei et al[2]: *FL with Differential Privacy*

- Introduces Gaussian noise w.r.t. a clipping parameter C to weight vector uploads, weights scaled w.r.t. C;
- Motivation: Naïvely uploaded weights carry the risk of being used by adversaries to compromise users;
- Proposes uplink and optional downlink noise addition;
- Takes fairly large values for the privacy budget;
- Accuracy improves with no. of clients queried and rounds of communication;
- Best accuracy when clients have (near) identical amounts of quality data;
- Akin to central differential privacy, straightforward noise addition.

Konečný et al[3]: FL:*Strategies for Improving Communication Efficiency*

- Introduces methods to reduce the uplink communication costs by reducing the size of the updated model sent back by the client to the server.
- Motivation: Poor bandwidth/expensive connections of a number of the participating devices (clients) leads to problems during aggregation of data for FL.
- *Structured updates*, updates are from a restricted space and can be parametrized using a smaller number of variables. Algorithms: Low Rank and Random Mask.
- *Sketched updates*, full model updates are learned but a compressed model update is sent to the server. Algorithms: Subsampling, Probabilistic quantization and structured random rotations.
- Some conclusions: Random mask $>$ low rank (lower size of updates), quantisation - more stable with rotation.

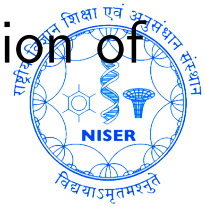
Some Algorithms/Techniques We Used

- For aggregation, we used `FedAvg` from [1] for the most vanilla FL implementation, and then added to it as necessary for the implementation of more sophisticated FL techniques.
- The Gaussian Mechanism to provide (ε, δ)-differential privacy pre-upload from each device (Noising before Aggregation), as introduced by Wei et al [2].
- *Static Sampling* of clients for every round of training to reduce communication rounds per user.
- *Random Mask* and *Probabilistic Quantisation* as described by Konečný et al in [3].
- *Dynamic Sampling* of clients to successively reduce the proportion of clients involved in successive rounds of communication to further save on rounds of communication and computational resources (as proposed by Ji et al [4]).
- Considering/considered using: *Selective Mask* from [4] (especially when working with larger weight vectors/matrices).

Experiments and Observations

Implemented `FedAvg` from scratch in Python, including a 0.25 static sampling rate, with anywhere between 3 to 5 rounds of calling for training from sampled (w.r.t. the uniform distribution) clients, with local training involving multiple linear regression implemented via Stochastic Gradient Descent on a synthetically generated dataset for about 500 clients, with static subsampling. The generated datasets, generated around a pre chosen/generated "true weights vector" (e.g. $[1, 2, 4, 3, 5, 6, 1.2]$), are mostly IID as we focused more on implementing a model atop that.

As we are using regression here, we use, for a particular instance, average training/testing error per data point as a metric of a deployment's accuracy, and simply measure the time taken by running it on Google Colab and compare them as a rough measure of how quick each is, and implement centralised learning to serve as a baseline for our exploration of these FL paradigms.



Centralised Learning

Throughout this entire exercise, we used a 7 dimensional weight vector. For centralised learning, we simply gathered and flattened the list of all local datasets into a cumulative list of all datapoints, and ran SGD on it for 100 epochs.
Time Taken $\approx 268 - 270$ seconds.
Average Training Error for Centralised Learning on SGD $\approx 3.2472 \times 10^{-29}$
Average Testing Error for Centralised Learning on SGD $\approx 3.2154 \times 10^{-29}$

Vanilla `FedAvg`

We then ran vanilla `FedAvg` with static sampling of clients at a rate of 0.25 of the client population per round of training on the above mentioned local datasets, with 100 epochs per round of local training. Number of rounds, $T = 5$.
Time taken ≈ 362 seconds.
Average Training Error for Vanilla FedAvg on SGD $\approx 2.5331 \times 10^{-18}$.
Average Testing Error for Vanilla FedAvg on SGD $\approx 2.6021 \times 10^{-18}$.

Noising before Aggregation[2]

N.B. Optionally adds downlink DP, but we felt it was unnecessary for most practical purposes.

Noising before Aggregation (NbA) FL with `FedAvg`

Ran `FedAvg` with static sampling of clients at a rate of 0.25 per round of training, clipping the locally generated weight vectors to gain bounds (defining upper bound $= 1.01 \times \max(w_i)$, where $w-$ weight vector), for the sensitivity of queries for the weight vectors, then adding Gaussian noise to each weight ($\varepsilon = 70, \delta = 0.1$) with 100 epochs per round of local training. Number of rounds, $T = 5$.
Time taken $\approx$ 349 seconds (as fast as vanilla `FedAvg`).
Average Training Error for NbA FedAvg on SGD $\approx 7.2562 \times 10^{-7}$.
Average Testing Error for NbA FedAvg on SGD $\approx 6.6740 \times 10^{-7}$.

NbAFL with Random Mask

Same setup as earlier, but with a layer of uniformly chosen random masks, excluding $\approx s = 0.25$ of the weights, prior to uploading by a client. Seems to consistently outperform base NbAFL in terms of accuracy, also reduces communication overhead!
Time taken $\approx$ 356 seconds.
Average Training Error ≈ 0.006601
Average Testing Error ≈ 0.007577

NbAFL with Probabilistic Binarisation

Same setup as NbAFL, but with probabilistic binarisation implemented atop it.
Time taken $\approx$ 356 seconds.
Average Training Error ≈ 0.021241.
Average Testing Error ≈ 0.019635

NbAFL with Dynamic Sampling

Same setup as NbAFL, but instead of static sampling of clients, we sample clients with an initial rate of 0.25, which decays by a factor of $\frac{1}{\exp(\beta t)}$, where t = number of rounds elapsed. We take $\beta = 0.05$.
Time taken $\approx$ 334 seconds.
(Saves on time and no. of communication rounds!)
Average Training Error ≈ 0.010684. Average Testing Error ≈ 0.010908

Name of Model	T	Client Sampling Rate	ε	*delta*	Training Error	Testing Error	Time Taken (sec)	Other Parameters
Centralised SGD	-	-	-	-	3.2472×10^{-29}	3.2154×10^{-29}	268 - 270	-
Vanilla FedAvg	5	0.25	-	-	2.5331×10^{-18}	2.6021×10^{-18}	362	-
NbAFL	5	0.25	70	0.01	7.2562×10^{-7}	6.6740×10^{-7}	349	-
NbAFL w/ Random Mask	5	0.25	70	0.01	0.006601	0.007577	356	$s = 0.25$
NbAFL w/ Prob. Bin.	5	0.25	70	0.01	0.021241	0.019635	356	-
NbAFL w/ Dyn. Samp.	5	0.25	70	0.01	0.010684	0.010908	334	$\beta = 0.05$

Table: Local Learning Rate, $\alpha = 0.01$; No. of local iterations for SGD $= 100$, No. of clients $= 500$

Errors - small (max. $\sim 1\%$ of ℓ_p norm of true weight vector)

Tentative Tweaks to Consider and Test Out

- Silo-ing users with similar characteristics and running a separate FL paradigm within these silos;
- Applications to dapps on P2P networks;
- Trying other mechanisms (viz. exponential) out on NbAFL for possible improvement;
- Trying Selective Mask from [4] out, and slightly diffusing the choice from the top k updates to some of the lower updates;
- Making dynamic sample more dynamic and efficient by calibrating decay w.r.t. error per round of communication (more error $\implies$ less decay;
- Exploring mutual benefits of probabilistic quantisation and noise addition, calibrating no. of quanta to the magnitude of the noise;
- Looking at tweaks to `FedAvg` in certain contexts;
- Anything else that strikes our minds.

We may tentatively try implementing the above FL models for more complex local training algorithms (viz. ConvNets using FEMNIST), but this is secondary/optional.

Bibliography

- H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data,", 2016.
- K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, 2020. DOI: 10.1109/TIFS.2020.2988575.
- J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *CoRR*, vol. abs/1610.05492, 2016. arXiv: 1610.05492. [Online]. Available: http://arxiv.org/abs/1610.05492.
- S. Ji, W. Jiang, A. Walid, and X. Li, "Dynamic sampling and selective masking for communication-efficient federated learning," *CoRR*, vol. abs/2003.09603, 2020. arXiv: 2003.09603. [Online].