CS460 Final Presentation

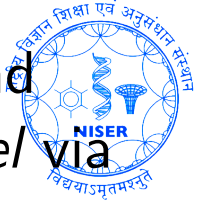
On Federated Learning and Possible Improvements

Saswat Das[1] Suraj Patel[2]

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

Our Post Midway Work

- Created a FL framework from scratch to transfer our work from one that employs local learning via linear regression on a randomly generated dataset to one that employs DNNs (implemented on Tensorflow) to classify digits in MNIST (randomly shuffled and distributed across clients).
- Tried out various approaches and tweaks to some existing paradigms/models.
- Designed "Adaptive Sampling" an improvement to Dynamic Sampling[3], with penalisation of the sampling decay coefficient for increase in errors.
- Designed a fully decentralised P2P approach to FL[4] (in contrast to the recent P2P models proposed that involve a level of temporary centralisation).
- Designed an approach involving clustering (of clients) in silos and simultaneously training a *silo-specific model* and a *general model* via FL.



Specifications of our new framework

Dataset: MNIST (training data shuffled and distributed unequally at random among 100 clients)

Local Training: Via a DNN, defined and compiled with Tensorflow.

- Input Layer: Flatten Layer.
- Hidden Layers: 32 units and 512 units (later ditched for speed) with ReLU activation.
- Dropout Layer (rate $= 0.2$): Before the 2^{nd} Hidden Layer (later ditched)
- Output Layer: 10 units with Softmax activation.
- Batch Normalisation: Before every hidden and output layer

Federated Averaging: Similar to previous implementation; tweaked to work for TF based NNs.

It is flexible enough so that with a handful of modifications, one can plug in any TF generated Neural Network with a relevant dataset.

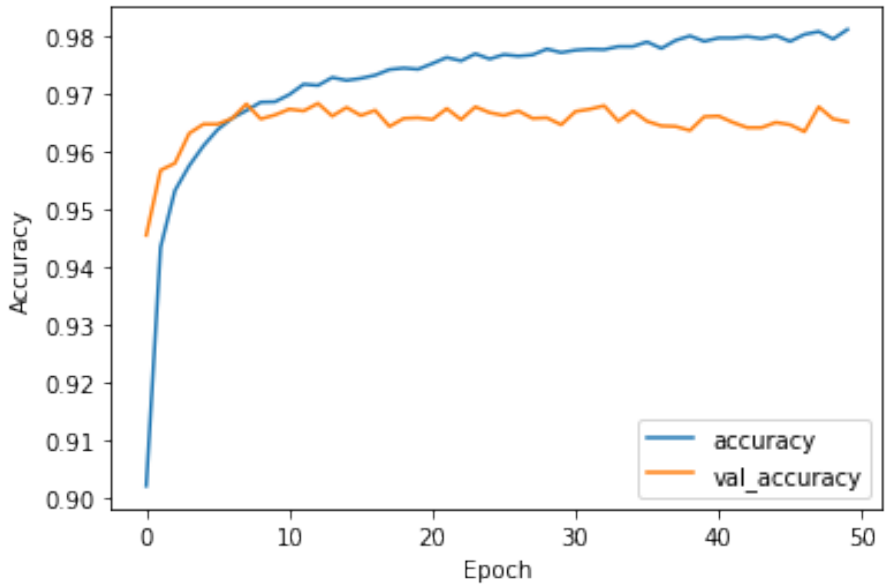
Baselines

N.B. For testing, we initialised the same server model for each run of each model, for a fair comparison.

Centralised Learning

i.e. train the DNN using all of the MNIST training data.

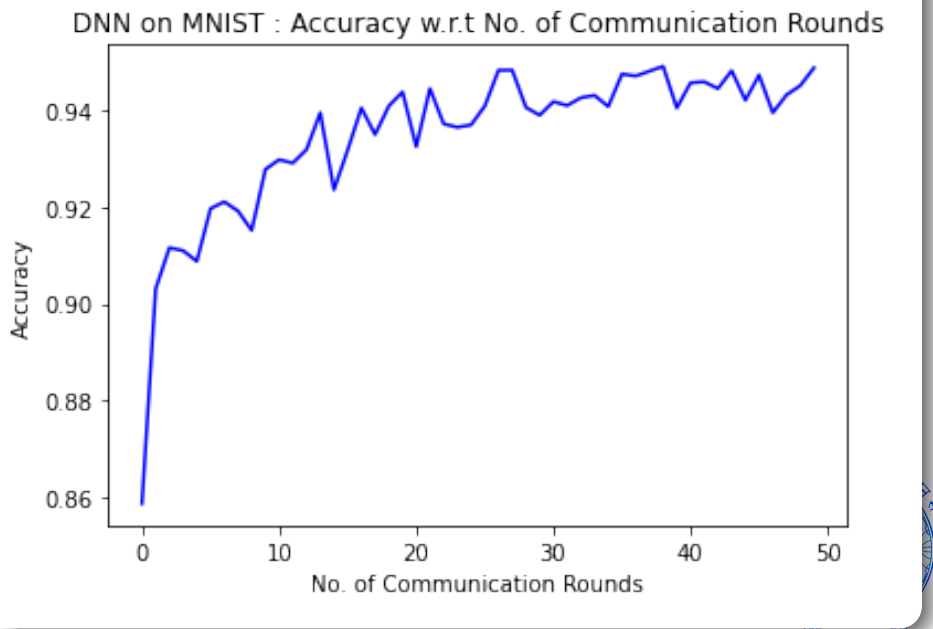
Test Accuracy: 96.52%



Vanilla FedAveraging

Number of Rounds: 50

Test Accuracy: 94.21%



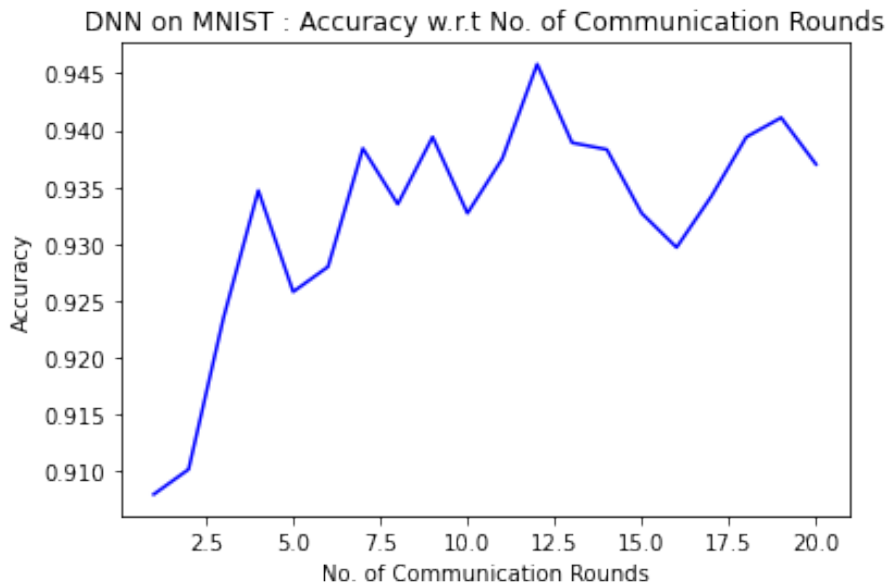
Baseline: Dynamic Sampling

Recap

- Proposed by Ji et al[3] in 2020.
- Instead of selecting (i.e. *sampling*) a fixed proportion C of clients at random per round, select $\frac{C}{\exp(\beta t)}$ fraction of clients, where C is the initial sampling proportion, and β is the decay coefficient (stays constant).
- Reduces communication cost as rounds progress, allows more number of federated rounds with the same communication cost
- Accuracy comparable to vanilla `FedAveraging`.

Dynamic Sampling

Decay Coeff. β: 0.05
Initial Sampling Rate : 0.25
Test Accuracy: 93.7%



Adaptive Sampling

Dynamic Sampling - decreases number of clients per round independent of the change of accuracy per round.

The problem? Can slow down convergence and error rectification, or even occasionally and briefly lead to consecutive increases in error. More clients involved $\implies$ Better averaging $\implies$ Less error.

Adaptive sampling
Keep penalising decay coefficient for fall in accuracy, while retaining the previous index, so as to increase the number of clients for the next round and slow decay down, and restore when accuracy reaches previous high.

Algorithm Adaptive Sampling

Require: $\beta \in \mathbb{R}_+, C, \gamma \in [0,1], T \in \mathbb{N}$ (No. of rounds) , $K \in \mathbb{N}$ (No. of clients)

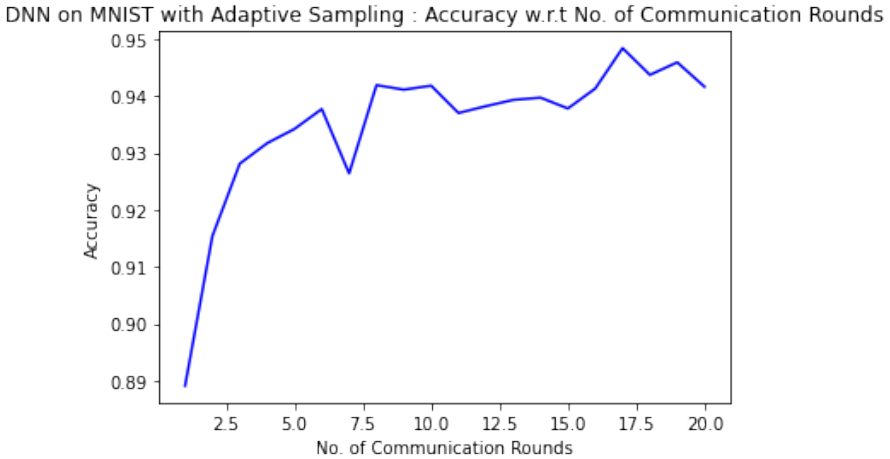
Central server: send initial model Θ_0 to clients and set decay coefficient β.
$t_0 \leftarrow 1, \Omega_{\text{thres}} \leftarrow 0$
for $t = 1 : T$, **do**
    Initialise empty list L
    $\beta_0 \leftarrow \beta$
    $c \leftarrow \frac{C}{\exp(\beta_0 t_0)}$
    $k \leftarrow \max(c \times K, 1)$
    Sample k clients and list them in S
    for $i \in S$, **do**
        Call ClientUpdate(Θ_0, i), which returns Θ_t^i
        L.append(Θ_t^i)
    end for
    Evaluate testing error Ω_t for $\Theta_t = \frac{1}{\text{len}(L)} \sum_{\Theta_t^i \in L} \frac{n_i}{n} \Theta_t^i$
    if $t > 0$ **then**
        if $\Omega_t > \Omega_{t-1}$ **then**
            $\beta_0 \leftarrow \beta_0 \times \frac{\Omega_{t-1}}{\Omega_t} \times \gamma$
            if $\Omega_{\text{thres}} = 0$ **then**
                $\Omega_{\text{thres}} \leftarrow \Omega_{t-1}$
            else
                $t_0 = t_0 + 1$
                if $\Omega_i \leq \Omega_{\text{thres}}$ **then**
                    $\beta_0 \leftarrow \beta$
                    $\Omega_{\text{thres}} = 0$
                end if
            end if
        end if
    else
        $i_0 \leftarrow 1$
    end if
end for
Output: $\Theta_{t+1} = \frac{1}{\text{len}(L)} \sum_{\Theta_t^i \in L} \frac{n_i}{n} \Theta_t^i$

Note : n_i is the number of datapoints possessed by the client corresponding to Ω_t^i, and n is the total of all n_i's for each $\Omega_t^i \in L$.

Adaptive Sampling

Adaptive Sampling

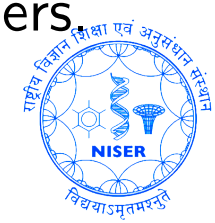
Initial Decay Coeff. β**:** 0.05
Initial Sampling Rate : 0.25
Gamma (γ) **:** 1
Test Accuracy: 94.16% (Would have risen as accuracy fell in the last step) (94.59% after step 19)



- More consistent in its accuracy growth than dynamic sampling!
- Reduces communication cost, though slightly less than dynamic sampling.
- Decrementally penalises β.
- Converges to a target accuracy value faster than dynamic sampling with just a slight increase in communication cost.

Baselines for P2P Learning

Federated learning in the context of P2P networks (i.e. sans a server) is a nascent field of study.
Some of the earliest models like BrainTorrent[6] (2019), and that by Behera et al[1] (JPMorgan Chase & Co., 2021) involve taking (i.e. randomly choosing/electing) a temporary "leader" peer node to act in the capacity of the server, thus inducing a level of centralisation, and the peers having to accept the leader's aggregation in good faith.
Others like FedP2P[2] (2021) involve a model wherein a central server organises clients into a P2P network and are not suitable for fully decentralised P2P networks, and models like IPLS[5] (2021) involve a client acting as a central "leader/server" that assigns tasks to its peers.



Fully Decentralised P2P FL Learning

Goal:

To introduce a completely decentralised FL paradigm for P2P networks, where clients (peer nodes) get to choose the model (local/shared global) that works best for them.

Algorithm:

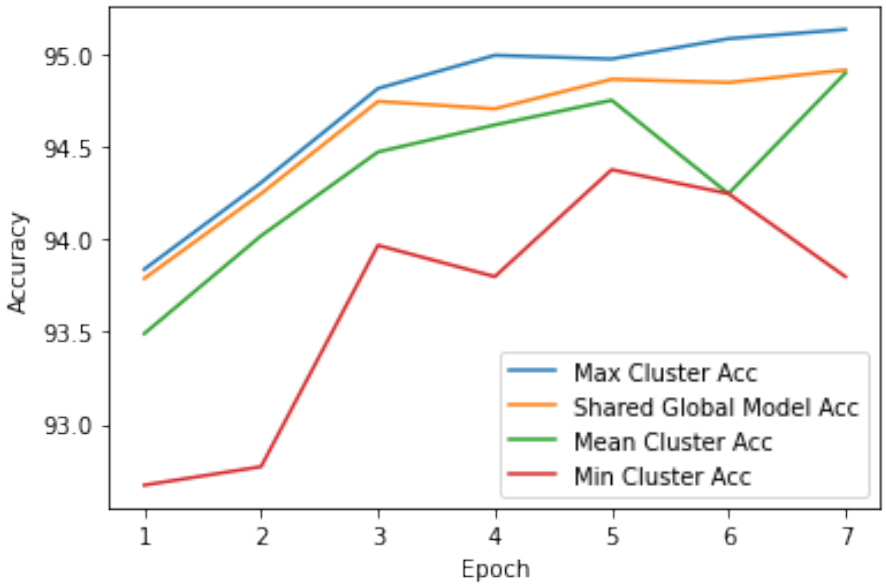
- The very first initial model is defined by, say, a smart contract or dapp.
- A device may send a call for updates to all its peers, periodically or of its own volition.
- If a good enough proportion of peers agree, then they initialise with a common initial model (specified within the contract initially, else the last shared global model), and train local models. Last 2-3 "accepted" global models are stored.
- Each device calls for a number of those locally trained weights from a significant proportion (or all) of its peers and aggregates it.
- If the model performs better than its existing model, it keeps it (updates local instance of dapp accordingly), else ditches it.
- Peers query for how many models ditched the new global model, if a majority do, then initialise the next round of training with the last viable one. (Defends the system against erroneous value injection.)

Fully Decentralised P2P FL Learning : Experiment Results

100 clients/peers, 0.25 sampling rate.

	Accuracy of Shared Global Model	Max Accuracy in Sample	Min Accuracy in Sample	Mean Accuracy in Sample
Round 1 (Initial)	93.79%	93.84%	92.67%	93.49%
Round 2	94.25%	94.31%	92.77%	94.02%
Round 3	94.75%	94.82%	93.97%	94.48%
Round 4	94.71%	94.99%	93.80%	94.62%
Round 5	94.87%	94.98%	94.38%	94.76%
Round 6	94.97%	95.09%	94.25%	94.85%
Round 7	94.92%	95.14%	93.80%	94.90%

Table: Results of Decentralised P2P Fed Learning



Shows overall upward trend!
Shared global model approaches baseline accuracy of vanilla FedAvg.

Fully Decentralised P2P FL Learning

Concerns:

- Large proportion of peer must agree for communication.
- Outliers may cause problems. (This is true for vanilla FedAvg anyway)
- Inter-peer bandwidth could be a bottleneck, so reduce communication size somehow.
- What about a malicious peer reporting incorrect weights? That's solved; inaccurate models thus aggregated are rejected by peers.

Possible augmentations/suggestions:

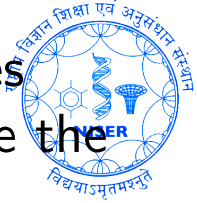
- Random mask can be used to reduce communication cost, .
- For security of communication, we can use hybrid encryption.
- For increasing privacy, differentially private noise can be added.
- To identify recurring malicious peer, devices rejecting the shared global model in each round may compare list of peers they got weights from. (Multiplicative weights based approach perhaps?) Refining this is an open problem.

Subclustering of Clients for Making Models for Minority Classes

Motivation: In general FL techniques make a model useful for majority of the devices, which is good but in this process a minor cluster of devices having different data than most may benefit from getting more specific models appropriate for their data.

Algorithm:

- We start with a regular FL (viz. Vanilla FedAvg) approach to training an FL model.
- Local models of all the devices are clustered (using DB-SCAN) based on their weight vectors.
- All the devices having the same cluster-id are extracted, and federated averaging is done only for those devices (might try this in a P2P manner). This process is repeated until a model is formed for every cluster.
- These weights are returned to the server to provide a model for devices having a different and smaller class of dataset. Devices may opt to use the general model or the specific model, if any.



Subclustering of Clients for Making Models for Minority Classes

N.B. For this, due to practical constraints with clustering algorithms (DB SCAN) for Tensorflow NN weights (even flattened), we ultimately ended up using the earlier framework with 7-dimensional weight vectors and multilinear regression.
Total number of devices = 125

		Average Testing Error	
	No. of Devices	For Global Model	For Cluster Specific Models
Majority Cluster	100	$\approx$0.00657	$\approx 2.52506\times10^{-28}$
Minority Cluster 1	8	$\approx$0.11422	$\approx 6.14606\times10^{-29}$
Minority Cluster 2	17	$\approx$0.08912	$\approx 5.08026\times10^{-29}$

Table: Results of Subclustered FL

Massive improvement within clusters!

Subclustering of Clients for Making Models for Minority Classes

Concerns:

- Needs more communication rounds than the usual models, i.e. a round each for global training and each of the specific models.

Possible future improvements:

- What about we store minority class models and use them as initial models for forks for the respective silos (given a minimum cluster size)? We leave it as an open problem.

Bibliography I

[1] Monik Raj Behera et al. *Federated Learning using Peer-to-peer Network for Decentralized Orchestration of Model Weights*. Mar. 2021. DOI: 10.36227/techrxiv.14267468.v1. URL: https://www.techrxiv.org/articles/preprint/Federated_Learning_using_Peer-to-peer_Network_for_Decentralized_Orchestration_of_Model_Weights/14267468/1.

[2] Li Chou et al. "Efficient and Less Centralized Federated Learning". In: *CoRR* abs/2106.06627 (2021). arXiv: 2106.06627. URL: https://arxiv.org/abs/2106.06627.

[3] Shaoxiong Ji et al. "Dynamic Sampling and Selective Masking for Communication-Efficient Federated Learning". In: *CoRR* abs/2003.09603 (2020). arXiv: 2003.09603. URL: https://arxiv.org/abs/2003.09603.

Bibliography II

[4] H. Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: (2016).

[5] Christodoulos Pappas et al. "IPLS : A Framework for Decentralized Federated Learning". In: *CoRR* abs/2101.01901 (2021). arXiv: 2101.01901. URL: https://arxiv.org/abs/2101.01901.

[6] Abhijit Guha Roy et al. "BrainTorrent: A Peer-to-Peer Environment for Decentralized Federated Learning". In: *CoRR* abs/1905.06731 (2019). arXiv: 1905.06731. URL: http://arxiv.org/abs/1905.06731.