Next Character Prediction Using Federated Learning Methods

Mohammad Sepehri
mohammad.sepehri@ut.ac.ir

1 Problem statement

In this project, I implement federated learning algorithms as one of the decentralized optimization methods you can find codes in this link: https://github.com/EMES888/Federated-learning.git

The objective of training a network is to find a set of parameters that minimize the cost function over the training set. Usually, these cost functions are not convex. However, to attain good generalization and performance on test datasets, you should have a large centralized datasets, which is not the case for many applications. Therefore, there is so much desire for decentralized optimization methods to solve this problem. This situation is usually called Federating learning; when we have decentralized datasets and want to use all the data, however, we do not or can not access the whole data for privacy(for example language models can improve speech recognition and text entry, and image models can automatically select good photos. However, this rich data is often privacy sensitive, large in quantity or both), communication cost, etc. In this case, the importance of a good decentralized optimization algorithm appears. In decentralization optimization which implemented in this project we focused on primal decomposition and dual decomposition. Primal and dual decomposition are two methods used in optimization to solve large-scale problems by breaking them down into smaller subproblems. Primal decomposition: In primal decomposition, a large-scale problem is decomposed into smaller subproblems that can be solved independently. The subproblems are typically formed by partitioning the variables in the original problem into groups. The objective is to find the optimal values of the variables that minimize the overall objective function subject to the constraints. In dual decomposition, the problem is decomposed by partitioning the constraints into groups. The subproblems are formed by considering each group of constraints and finding the optimal dual variables that satisfy those constraints. The objective is to find the optimal values of the dual variables that maximize the overall objective function subject to the constraints. The main difference between primal and dual decomposition is the way the problem is decomposed. In primal decomposition, the variables are partitioned, whereas in dual decomposition, the constraints are partitioned. The choice between primal and dual decomposition depends on the problem at hand and the structure of the constraints and variables. In dual decompostion there are some issues like our cost function need to be strictly convex to retrive primal variables and converge and our Lagrangian function should be down limited and problem should be decomposed for calculate decomposition solution so another method introduced which called *Alternating Direction Method of Multipliers*(ADMM)(Boyd et al., 2011). In conventional federated learning, at every step, all local clients update their parameters in parallel and them to central server for aggregation when ADMM is applied into federated learning, it can be viewed as a scene where the local clients update not only their parameters but also the dual parameters of the target optimization problem. The *Federated Averaging*(McMahan et al., 2016) is one of the decentralize optimization algorithms which is used primal method. *FedProx*(Sahu et al., 2018) is another primal manner. Some algorithms with ADMM based are *ADMM-based Federated learning*(Zhou and Li, 2021), *LIADMM: Linearised inexact ADMM-based federated learning*(Zhou and Li, 2021) and *FedADMM*(Gong et al., 2022). While we focus on non-convex neural network objectives, the algorithms we consider is applicable to any finite-

sum objective of the form

$$minf(w)$$

where

$$f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w)$$

for a machine learning problem, we typically take $f_i(w) = l(x_i, y_i; w)$, that is the loss of the prediction on example (x_i, y_i) made with model parameters w. We assume there are K clients over which the data is partitioned, with ρ_k the set of indexes of data points on client k, with $n_k = \rho_k$. Thus we can re-write the above objective as

$$f(w) = \sum_{k=1}^{k} \frac{n_k}{n} F_k(w)$$

where

$$F_k(w) = \frac{1}{n_k} \sum_{i \in \rho_k} fi(w)$$

Federated optimization has some key properties that differentiate it from a typical distributed optimization problem like *Non-IID* distribution of data between clients, The training data on a given client is typically based on the usage of the mobile device by a particular user, and hence any particular user's local dataset will not be representative of the population distribution. In this work, our emphasis is on the non-IID and IID properties of the optimization, but critical nature of the communication constraints is more close to non-IID. In all part of this project IID and non-iid is in a same way for different models to have a better approach for comparison and number of clients for all models is 10 and in IID length of data for each client is same and for non-IID manner each client get different percent of whole data so by this way we can have a non-iid distribution.

2 What we proposed and Accomplishments

In Proposal we mentioned that we want to implement federated learning methods like FedAVG which is try to solve decomposition problem in next character prediction task but in this project we try to implement some algorithms like FedProx, Admm based federated learning, Linearized inexact federated learning and Fedadmm and compare them on both IID and Non-IID distributions of data in clients.

3 Related work

Recent works on language modelling in Federated NLP mainly target on solving a word-level LM problem in mobile industry. That is mobile keyboard suggestion, which is a well representative of federated NLP applications. To improve mobile keyboard suggestions, federated NLP models aim to be more reliable and resilient. Existing models offer quality improvements in typing or even expression (e.g., emoji) domains, such as next-world predictions, emoji predictions, query suggestions, etc. Considering the characteristics of mobile devices, a decentralized computation approach is constrained by computation resource and low-latency requirement. A mobile device has limited RAM and CPU budgets, while we expect keyboards to provide a quick and visible response of an input event within 20 milliseconds. Thereby, the model deployed in client sides should perform fast inference. Most works (Hard et al., 2018; Yang et al., 2018; Ji et al., 2018; Chen et al., 2019; Ramaswamy et al., 2019; Stremmel and Singh, 2020; Thakkar et al., 2020) consider variants of LSTMs as the client model. Given the limited computation budget on each device, we expect the parameter space of a neural language model to be as small as possible without degrading model performance.CIFG (Greff et al., 2015) is a promising candidate to diminish LM's complexity and inference time latency. It employs a single gate to harness both the input and recurrent cell self-connections. FedProx (Sahu et al., 2018) allows for more robust convergence that FedAvg across a suite of sealistic federated datasets.

4 Dataset

Data were obtained from the Gutenberg project which has all Shakespeare play compiled into a single file. This text file is 12,788 lines, 1,134,708 tokens, 5,465,129 characters and 91 unique characteres.

4.1 Data preprocessing

To begin with, it is necessary to remove pieces unnecessary texts such as information about the Gutenberg project, terms of use license and poems. This data cleansing will help to get text with greater predictive value for the next-character on into Shakespeare's plays. Natural Language Processing (NLP) techniques for data processing will be applied like remove stop words and tokeniza-

tion. A bag-of-words (BoW) is a model maps a document into a vector as $v = [x_1, x_2, ..., x_n]$, where x_i denotes the occurrence of the ith word in basic terms . In this case a BoW was generated in resulting a sparse array of data. To better visualize the BoW, only the twenty most common tokens in all parts were keep. Term Frequency and Inverse Document Frequency (TF-IDF) determine the importance of a feature in a text document. In this approach, instead of filling the document vectors with the raw count, as in the BoW approach, it is filled with a score. This technique comes in very handy for the next word prediction problem because it calculates the rarity of a word in a text.

4.2 Data Exploration

Using the BoW technique, the first twenty unigrams and bigrams of the text were calculated and how result, the figures 1 and 2 generated.

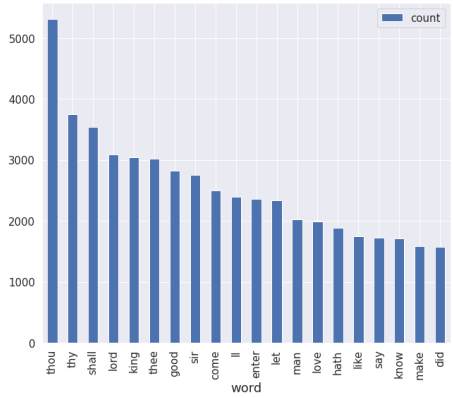

Figure 1: The distribution of top unigrams after data cleansing.

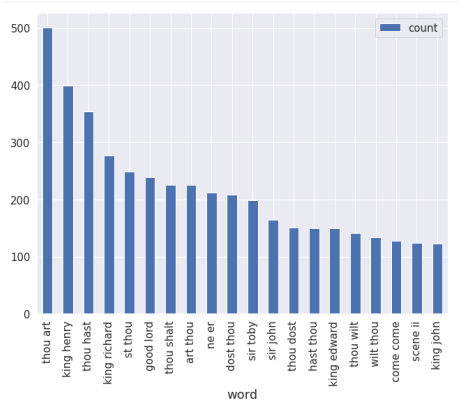

Figure 2: The distribution of top bigrams after data cleansing.

5 Baselines

For the Baseline Model I used LSTM because of simplicity of these models and can compare with decentralized models, and we don't seek for getting a high accuracy we try to compare centralize and decentralize methods. We are mainly focused on how learning happend in different type of modeling and compare centralize and decomposed methods and as see results in related papers we mainly focused on training and not evaluation on unseen data. Our baseline LSTM we defined 100 as sequence length and give data to model as batches and size of batch is 10 and number of layers is 2 and hidden size is 256 and *Cross Entropy Loss* as our loss function and *Adam* as our Optimizer and you can see result of training during 20 epochs 3

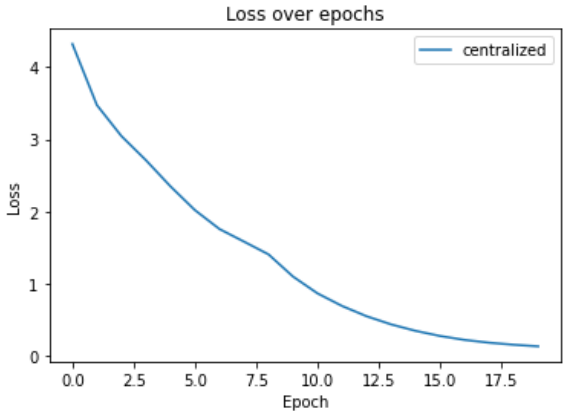

Figure 3: Loss changes in LSTM model in 20 epochs

6 Approach

6.1 Federated Averaging

In Federated Averaging(FedAvg)(McMahan et al., 2016) the local surrogate of the global objective function at device k is F and the local solver is Adam with the same learning rate and number of local epochs used on each device. at each round a subset of the total devices are selected and run Adam locally for E number of epochs, and then the resulting model updates are averaged. we have a 10 client and in IID manner our data distribute to each client equally and in non-IID manner we have different length of data for each client. and model of each client is LSTM and Cross Entropy loss as loss function and Adam as optimizer our sequence length is 100 and embedding dimention is 64 and hidden size is 256 and each client run

in 10 epoch with batch size of 10 and 0.001 as learning rate and we have 10 round over all clients to update global model. The details of *FedAvg* are summarized in algorithm 1.

Algorithm 1 Federated Averaging (FedAVG)

Require: Number of clients K, local epochs E, learning rate η, and server iteration number t

Initialize the model parameters w_0
for each iteration $t = 1, 2, \ldots, T$ **do**
  $m_t \leftarrow \max(C \cdot K, 1)$ ▷ Select $C \cdot K$ clients at random
  $S_t \leftarrow$ (randomly partition the data of the m_t clients into K non-empty sets)
  for each client $k \in S_t$ in parallel **do**
    $w_{t,k} \leftarrow w_{t-1}$ ▷ Initialize the client model parameters
    for each local epoch $e = 1, \ldots, E$ **do**
      $w_{t,k} \leftarrow w_{t,k} - \eta \nabla \ell(w_{t,k}; \mathcal{D}k)$ ▷ Update client model parameters using local data
    end for
  end for
  $w_t \leftarrow \frac{1}{m_t} \sum k = 1^{m_t} w_{t,k}$ ▷ Aggregate client models at the server
end for
return the final model w_T

6.2 FedProx

Our other primal is *FedProx*(Sahu et al., 2018). FedProx, is similar to FedAvg in that a subset of devices are selected at each round, local updates are performed, and these updates are then averaged to form a global update. However, FedProx makes the following simple yet critical modifications, which result in significant empirical improvements and also allow us to provide convergence guarantees for the method. Our parametrs in Fedprox is similar to FedAvg parameters and μ in this algorithm is set to 0.5. in this algorithm we We propose to add a proximal term to the local subproblem to effectively limit the impact of variable local updates. In particular, instead of just minimizing the local function F_k , device k uses its local solver of choice to approximately minimize the following objective F_k:

$$minh_k(w; w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|$$

see pseudocode in 4. You can see results of both FedProx and FedAVG in 7,8,5,6. As you can see

Algorithm 2 `FedProx` (Proposed Framework)

Input: $K, T, \mu, \gamma, w^0, N, p_k, k = 1, \cdots, N$
for $t = 0, \cdots, T-1$ **do**
  Server selects a subset S_t of K devices at random (each device k is chosen with probability p_k)
  Server sends w^t to all chosen devices
  Each chosen device $k \in S_t$ finds a w_k^{t+1} which is a γ_k^t-inexact minimizer of: $w_k^{t+1} \approx \arg\min_w h_k(w; w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$
  Each device $k \in S_t$ sends w_k^{t+1} back to the server
  Server aggregates the w's as $w^{t+1} = \frac{1}{K}\sum_{k \in S_t} w_k^{t+1}$
end for

Figure 4: Fedprox alg.

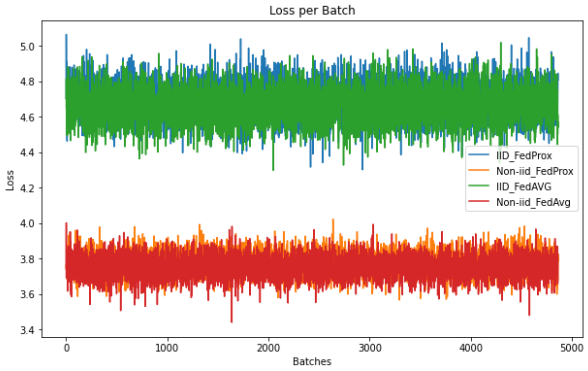

Figure 5: Loss of FedAVG and FedProx during batches.

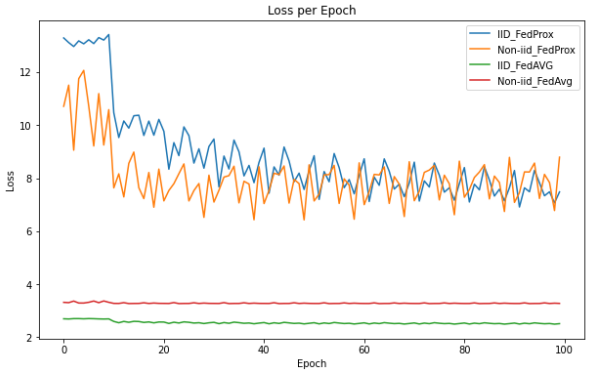

Figure 6: Loss of FedAVG and FedProx during epochs.

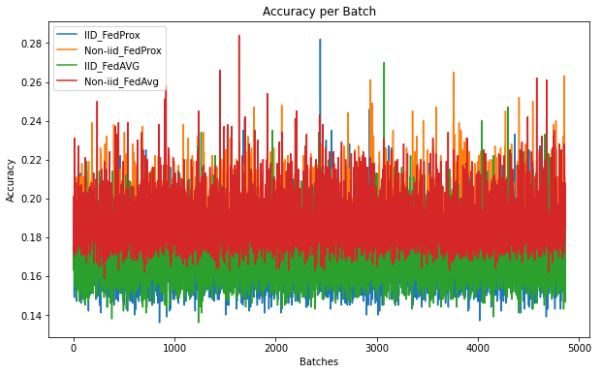

Figure 7: Accuracy of FedAVG and FedProx during batches.

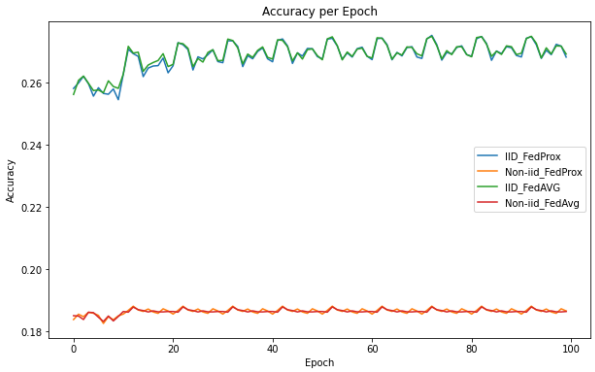

Figure 8: Accuracy of FedAVG and FedProx during epochs.

in results of both FedAVG and FedProx it seems both of them trat similar but loss term in FedAVG seems to get better results and we should noticed that in(Sahu et al., 2018) noticed that Fedprox can gets better results in longer training, and you can see both algorithms get better results in IID distribution between clients.

6.3 Linearised inexact ADMM-based

We first briefly introduce some backgrounds of the alternating direction method of multipliers (ADMM). For more details, one can refer to a nice book(Boyd et al., 2011). Suppose we are given an optimization problem,

$$minf(x) + g(z)$$

$$s.t.$$

$$Ax + Bz = b$$

To implement ADMM, we need the augmented Lagrange function of problem, which is defined by

$$l(x, z, \pi) := f(x) + g(z) + (Ax + Bz - b, \pi) +$$

$$\frac{\rho}{2}\|Ax + Bz - b\|$$

where π is known for Lagrange multiplier and $\rho > 0$. Based on the augmented Lagrange function, ADMM executes the following steps for a given starting pointx^0, z^0, π^0 and any $k >= 0$

$$x^{k+1} = argminL(x, z^k, \pi^k)$$

$$z^{k+1} = argminL(x^{k+1}, z, \pi^k)$$

$$\pi^{k+1} = \pi^k + \rho(Ax^{k+1} + Bz^{k+1} - b)$$

Therefore, to implement ADMM for our problem in federated learning the corresponding augmented lagrange function can be defined by

$$L(x, X, \pi) := \sum_{i=1}^{m} L(x, x_i, \pi_i)$$

, where $X := (x_1, x_2, \cdots, x_m)$, $\Pi := (\pi_1, \pi_2, \cdots, \pi_m)$, and $L(x, x_i, \pi_i)$ is:

$$L(x, x_i, \pi_i) := w_i f_i(x_i) + (x_i - x, \pi_i) + \frac{\rho_i}{2}\|x_i - x\|^2$$

Here π_i are the lagrange multipliers and $\rho_i > 0$ similar to ADMM formulation we have the framework of ADMM for problem of federated learning. That is for an initialized point$(x^0, X^0, \Pi^0$ and

any $k >= 0$ preform the following updates iteratively:

$$x^{k+1} = argminl(x, X^k, \Pi^k)$$
$$x_i^{k+1} = argminL(x^{k+1}, x_i, \pi_i^k)$$
$$\pi_i^{k+1} = \pi_i^k + \rho_i(x_i^{k+1} - x^{k+1})$$

You can see pseudocode algorithm in 9. After seeing the ADMM based algorithm you can see each client i needs to calculate two parameters x_i^{k+1} and π_i^{k+1} after receiving global parameter y^{k+1}, and solving former problem for calculating our global parameter(y) generally does not admit a closed-form solution, thereby leading to expensive computational cost. To accelerate the computation for local clients, many strategies aim to solve subproblem approximately. A common approach to find an approximate solution for our problem takes advantage of the second order Taylor-like expansion(Zhou and Li, 2021).and this algorithm called Linearised inexact ADMM-based federated learning Our hyperparameters in implementation are: sequence length=100 and step size=0.001 and each client run 10 epoch and 10 times we go train over clients and we have cross entropy loss for loss function and Adam as optimizer. and each client has a LSTM network for doing task and you can see pseudocode of algorithm in 10. You can see results of this algorithm in 11,12 As you can see in

Algorithm 2: ADMM-based federated learning

Initialize $\mathbf{x}_i^0, \boldsymbol{\pi}_i^0, \sigma_i > 0, i \in [m]$. Set $k \Leftarrow 0$.

for $k = 0, 1, 2, \ldots$ **do**

Weights upload: Each client sends the parameters $\mathbf{x}_i^k$ and $\boldsymbol{\pi}_i^k$ to the central server.

Global aggregation: The central server calculates the average parameter $\mathbf{x}^{k+1}$ by

$$\mathbf{x}^{k+1} = \text{argmin}_{\mathbf{x}} \mathcal{L}(\mathbf{x}, X^k, \Pi^k). \quad (2.16)$$

Weights feedback: The central server broadcasts the parameter $\mathbf{x}^{k+1}$ to every local client.

for $i = 1, 2, \ldots, m$ **do**

Local update: Each client updates its parameters locally and in parallel by

$$\mathbf{x}_i^{k+1} = \text{argmin}_{\mathbf{x}_i} L(\mathbf{y}^{k+1}, \mathbf{x}_i, \boldsymbol{\pi}_i^k),$$
$$\boldsymbol{\pi}_i^{k+1} = \boldsymbol{\pi}_i^k + \sigma_i(\mathbf{x}_i^{k+1} - \mathbf{y}^{k+1}).$$

end

end

Figure 9: Admm-base alg.

Liadmm results by training in non-iid we get better results in shorter training and smother curves but for concluding better I think we should train more and because training this algorithm takes while we train it on fewer epochs but still in non-iid manner it can shows itself.

6.4 FedADMM

In FedAdmm(Gong et al., 2022) algorithm they combine primal and dual variables($w_i + \frac{1}{\rho} y_i$ which

Algorithm 1 `FedADMM`

Input: Total number of rounds T, server step size η, proximal coefficient ρ.
for $t = 0, 1, \ldots, T-1$ **do**
  Server selects $S^t \subset [m]$
`Clients`: *// In parallel*
  for $i \in S^t$ **do**
    download θ^t from the server
    $(w_i^{t+1}, y_i^{t+1}) \leftarrow$`ClientUpdate`$(i, \theta^t)$
    compute update message Δ_i^t as in (4)
    upload Δ_i^t to the server
  end for
`Server`:
  $\theta^{t+1} \leftarrow \theta^t + \frac{\eta}{|S^t|} \sum_{i \in S^t} \Delta_i^t$
end for

`ClientUpdate`(i, θ): *// Store w_i and y_i*
Input: Local epoch number E_i, client learning rate η_i.
create batches $\mathcal{B}$
for $k = 0, 1, \ldots, E_i - 1$ **do**
  for batch $b \in \mathcal{B}$ **do**
    compute batch gradient $\nabla f_i(w_i, b)$
    $w_i \leftarrow w_i - \eta_i \big(\nabla f_i(w_i, b) + y_i + \rho(w_i - \theta)\big)$
  end for
end for
$y_i \leftarrow y_i + \rho(w_i - \theta)$
return w_i, y_i

Figure 10: Fedadmm alg.

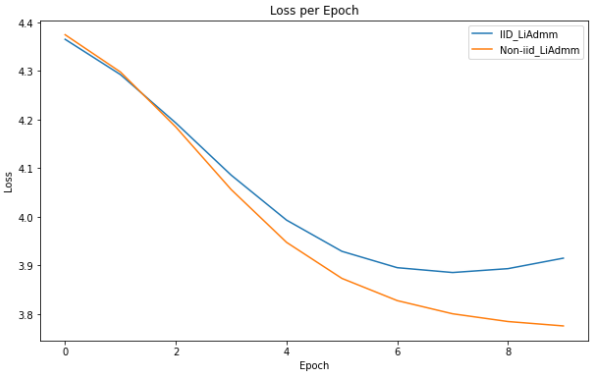

Figure 11: Loss of Liadmm during epoch.

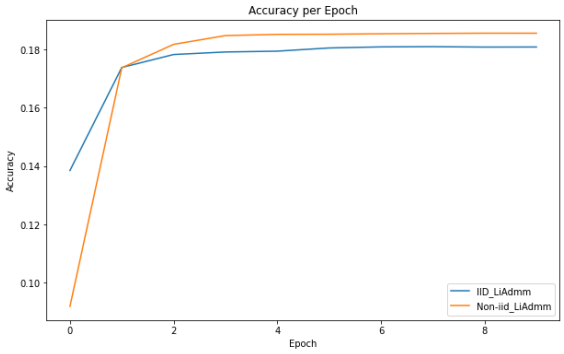

Figure 12: Accuracy of Liadmm during epoch.

w_i refers to x_i in Liadmm and use Δ_i^t to denote the update message of client i to the server, that is the difference between successive augmented models:

$$\Delta_i^t = (w_i^{t+1} + \frac{1}{\rho} y_i^{t+1}) - (w_i^t + \frac{1}{\rho} y_i^t)$$

The server updates the global model after gathering update messages from selected clients as:

$$\theta^{t+1} = \theta^t + \frac{\eta}{S^t} \sum_{i \in S^t} \Delta_i^t$$

and hyperparameters are same as Liadmm part so we can have a better compare and you can see pseudocode of algorithm in appendix. You can see results of both FedADMM an LiAdmm to compare them in 13,14.

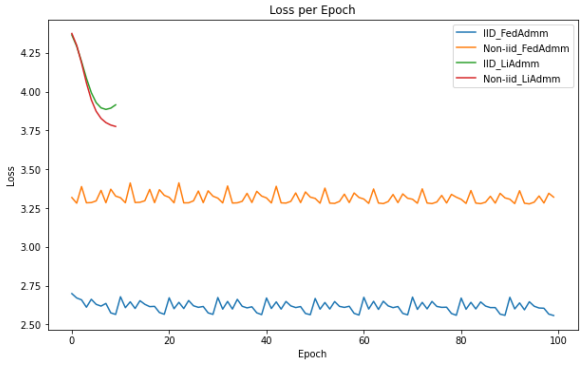

Figure 13: Loss of Fedadmm and Liadmm during epochs.

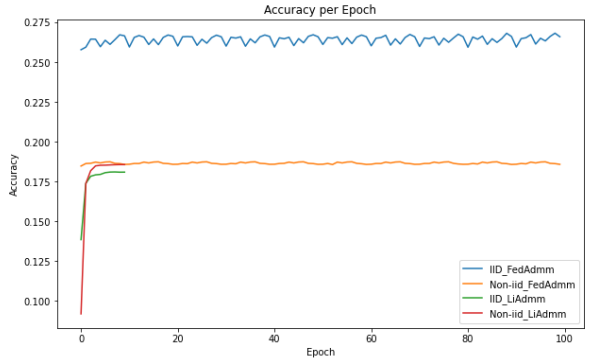

Figure 14: Accuracy of Fedadmm and Liadmm during epochs.

As you can see in in FedAdmm we get better result in IID and in total FedAdmm gets better results rather than Liadmm but in Liadmm we train on fewer epochs but we get good result on non-IID distribution.

7 Conclusion and Analysis

To compare all of our models you can see figures 15 16 17 18 As we can see in all results figures

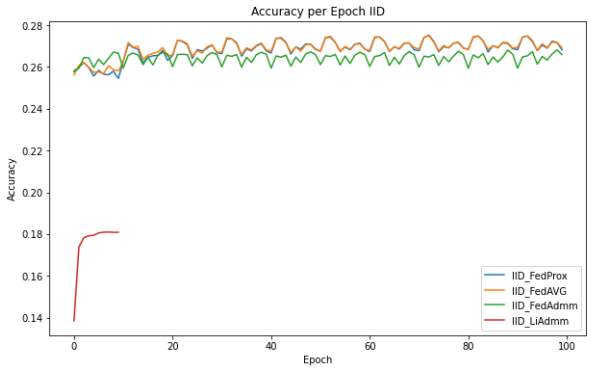

Figure 15: Accuracy of all models in IID manner.

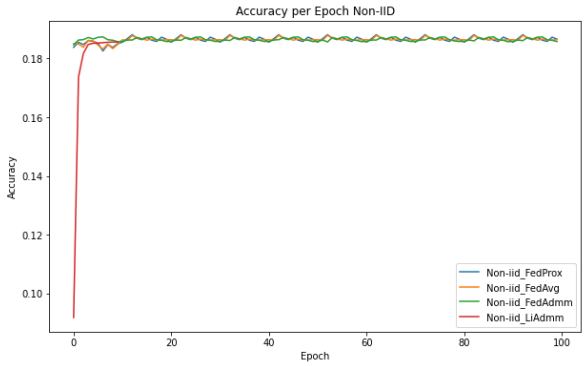

Figure 16: Accuracy of all models in non-IID manner.

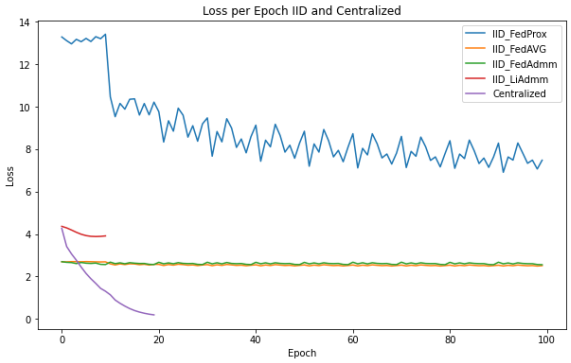

Figure 17: All model losses with IID distribution and Centralized manner.

you can see in most of models we get better results in IID manner except LiAdmm which we get better result in non-iid manner but we should check to see is this difference is statistically significant or not and in iid manner Liadmm is not as good as others but in non-iid is close enough. and after see models result we can see FedAVG and FedProx and FedAdmm are close but in non-iid Fedprox loss get higher loss but in accuracy are close but in total our centralize manner get best result and

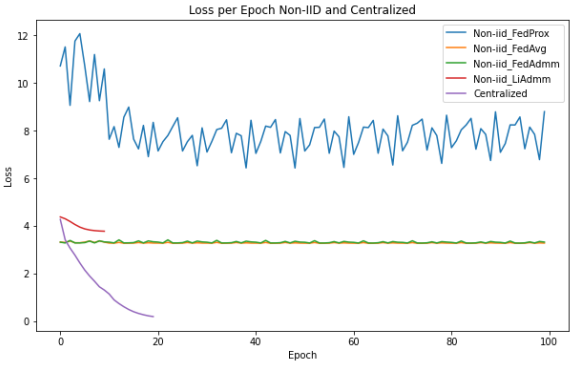

Figure 18: All model losses with non-IID distribution and Centralized manner.

our models have distance from it but it is logical because we solve our next character prediction in decomposition manner and results are reasonable enough. So we try to get close to centralize result but in decompositon methods like federated learning it is our goal but because of privacy and other reasons centralize method isn't always in our hand but with these algorithms we deal with data that isn't centralize.

References

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011).

Chen, M., Mathews, R., Ouyang, T., and Beaufays, F. (2019). Federated learning of out-of-vocabulary words. *CoRR*, abs/1903.10635.

Gong, Y., Li, Y., and Freris, N. M. (2022). Fedadmm: A robust federated deep learning framework with adaptivity to system heterogeneity.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2015). LSTM: A search space odyssey. *CoRR*, abs/1503.04069.

Hard, A., Rao, K., Mathews, R., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., and Ramage, D. (2018). Federated learning for mobile keyboard prediction. *CoRR*, abs/1811.03604.

Ji, S., Pan, S., Long, G., Li, X., Jiang, J., and Huang, Z. (2018). Learning private neural language modeling with attentive aggregation. *CoRR*, abs/1812.07108.

McMahan, H. B., Moore, E., Ramage, D., and y Arcas, B. A. (2016). Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629.

Ramaswamy, S., Mathews, R., Rao, K., and Beaufays, F. (2019). Federated learning for emoji prediction in a mobile keyboard. *CoRR*, abs/1906.04329.

Sahu, A. K., Li, T., Sanjabi, M., Zaheer, M., Talwalkar, A., and Smith, V. (2018). On the convergence of federated optimization in heterogeneous networks. *CoRR*, abs/1812.06127.

Stremmel, J. and Singh, A. (2020). Pretraining federated text models for next word prediction. *CoRR*, abs/2005.04828.

Thakkar, O., Ramaswamy, S., Mathews, R., and Beaufays, F. (2020). Understanding unintended memorization in federated learning. *CoRR*, abs/2006.07490.

Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong, N., Ramage, D., and Beaufays, F. (2018). Applied federated learning: Improving google keyboard query suggestions. *CoRR*, abs/1812.02903.

Zhou, S. and Li, G. Y. (2021). Communication-efficient admm-based federated learning.