DS603: Privacy-Preserving Recommendation Model

Aman Vishnoi

***Abstract*—Abstract—The main aim of this project is to create a privacy preserving federated news recommendation system. News recommendation aims to display news articles to users based on their personal interest. Existing news recommendation methods rely on centralized storage of user behavior data for model training, which may lead to privacy concerns and risks due to the privacy-sensitive nature of user behaviors. We will try to develop a privacy- preserving method for news recommendation model training based on federated learning, where the user behavior data is locally stored on user devices. The dataset that we will be using in the project is Mind Dataset. The MIND dataset for news recommendation was collected from anonymized behavior logs of Microsoft News website by randomly sampling 1 million users who had at least 5 news clicks during 6 weeks from October 12 to November 22, 2019. We will be trying out several new ideas in the paper as incorporating differential privacy, monitoring the model performance against federated attacks(model poisoning, Substitution-Based Profile Pollution Attacks), testing the model against several new benchmarks other than news as songs dataset, implementing of fairness-aware Federated Matrix Factorization**

I. Introduction

Many news recommendation methods still rely on centralized storage of user behavior data for model training, which raises some concerns regarding data privacy of user[1]. In this paper the user's data will be stored on the user's device.This technique can leverage the useful information in the behaviors of massive number users to train accurate news recommendation models and meanwhile remove the need of centralized storage of them. We keep a copy of small user model on each edge device and gradients are pushed to server where a large news model is being trained. Since we will be taking gradients from local devices we will be using techniques as Multiparty computation, Local Differential Privacy for privacy protection. The updated global model is then distributed to each user device for local model update. We repeat this process for multiple rounds. We will be monitoring the performance of our model against various attacks as substitution-based profile pollution attacks[2], and model poisoning[3]. As Fairness and robustness are two important concerns for federated learning systems[4] we have also incorporated several implementations that will make the model fair.

II. RelatedWork

Work have already been done in federated recommendation system by Tao Qi, Fangzhao Wu,Chuhan Wu, Yongfeng Huang, Xing Xie in his paper Privacy-Preserving News Recommendation Model Learning[1], and they are using multiparty computation, instead of diffrential privacy or homomorphic encryption. The model has not been tested to attacks and is not tested for fairness. Another approach has been taken up Tao Qi, Fangzhao Wu, Chuhan Wu, Yongfeng Huang, and Xing Xie in their paper Privacy-preserving news recommendation model learning[1]. However, the communication and computation cost of FedRec is unacceptable for user devices with limited resource due to the large size of news recommendation models, especially their news models.

III. SystemModel

The models have been trained on MIND and address data on P5000 GPU, 30 GB ram and 16 core CPU for a period of 8 hours and the logs have been published over wandb. Addressa dataset have to preprocessed into MIND dataset format to be used in the model.

IV. Algorithms

The first step is to properly format the news in dataset. We have dataset that contains user's history, the news he have clicked and the news he ignored. We will be denoting news clicked as a positive sample and rest all news as a negative sample. We will be creating a dataloder that will return an array containing a positive sample, all negative samples, history for a particular user. For the purpose of training we will be randomly sampling 50 users and aggregate their history, their positive and negative samples via Multiparty computation. We will be using BERT model pretrained model to generate news vector embeddings of 400 dimensions for the entire MIND/addressa dataset. We will be unfreezing some of pretrained layers in middle of BERT for fine tuning over news dataset. We have a user-encoder model in the server and one at each client devices. The user-model has multihead attention layer connected to two linear layers which when fed the encoding of user's history generates a user-embedding of 400. We then create a batch of news history from the randomly sampled users and generate user encoding for each user by the user's model stored at server. We then create encoding of news that we are trying to do the prediction over, i.e. positive samples and negative samples. We will then do dot product for these samples and user-vector to get the scores and then we will apply softmax, since we know the positive sample and negative sample we will do will do cross entropy loss for each of these users to do back-propagation to update BERT model and user models at the server. The weights of this user model at the server is sent back to each of the clients. Then again again randomly selects 50 user and again do backpropagation for the user model at the server for the news of these 50 people and send the gradients back to the user model at the client.

V. Experiments and results

The prime dataset that I will be using are Mind Dataset for news articles, and addressa dataset. The result for MIND

Fig. 1: Addressa dataset

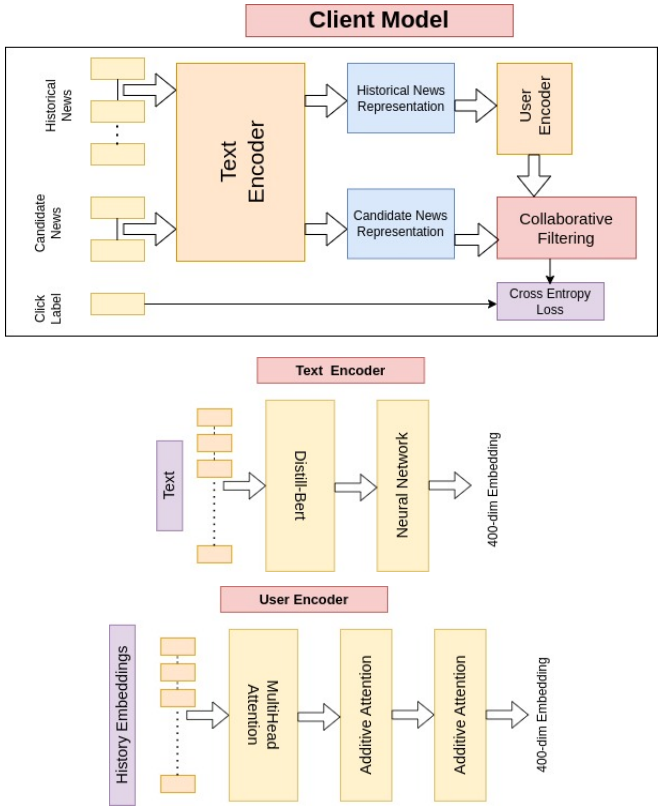
Fig. 2: Mind dataset

Dataset have been replicated, however the reported MRR and NDCG for addressa in the paper is not correct according to the experiments conducted by me.

1) **Mind Dataset**
2) **Addressa Dataset**

Experiment Results				
Dataset	MRR	AUC	NDCG@5	NDCG@10
MIND	32.86	68.42	36.43	42.62
Addressa	37.67	72.04	37.33	45.31

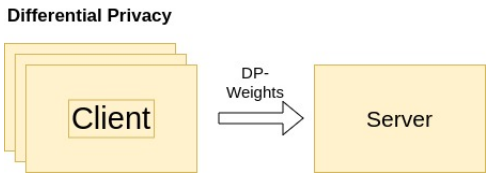
VI. Enhancement done in the paper



I have written down all the code from scratch with the help of original repository. I have done some changes in the client architecture. The client now, instead of news representation like in early architecture, have two models, one of them is user-encoder and other one of them is text-encoder. Text-Encoder uses Distill-Bert Architecture instead of Bert Encoder, which enable us to cut down the parameters by 40% . Since I have frozen the Distill Bert for training I have added a neural architecture in front of Distill Bert which is open for training so that the text encoder can adapt itself to a particular user and gives me an embedding of dimensions 400. User Encoder takes in the embedding of history of the user and apply additive attention(custom architecture used in the original paper to cut down n-dim embedding into 1-dim embedding), followed by a vanilla neural network that allows us to obtain an embedding of 400 dimension. Then given the user-encoding and the candidate news encoding, I apply collaborative filtering followed by a cross-entropy loss function to train the entire network. Since the parameters have been cut down and user device has a small chunk of data, the entire model can be trained on a CPU with 4GB memory requirement.

The client will then be sending the weights of the model trained to the server. For ensuring the safety of client's data I have added the concepts of Differential Privacy for training. I have added Gaussian noise $(\epsilon = 10, \delta = 10^{-}5)$ in the gradient obtained while dealing with mini-batches, thus in case of back propagation individual data-samples of client are

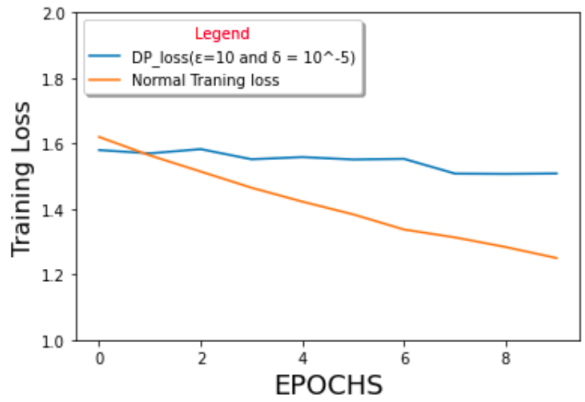
protected.The overall structure looks equivalent to something like-



A. Issues while creating the client architecture

1) There is not enough data available for a particular client in the mind data-set. For some user there is training data in the training folder, but there is no validation and testing data. So I need to create new dataset which will basically be the subset of MIND dataset for those users who have training, testing and validation data available.
2) Very few training data is available even for a particular client. This is normally leading to over fitting which might affect the performance of model on the client's side.
3) The convergence of the entire network will be a lot slower since in the initial phase server has learnt nothing, a better approach will be give some data to server model and when server send weights to client, client will atleast have something to start with thus enabling faster convergence.
4) I have myself added Gaussian noise in the parameters gradient and I have not done gradient clipping. To optimize code and handle the edge cases I need to implement opacus.

B. Training Plots



The following figure plots the training loss with the number of epochs in the client's side. As we can see that diffrential privacy affects accuracy, so we need to carefully model the variation of ϵ with the training loss.

C. Future Work

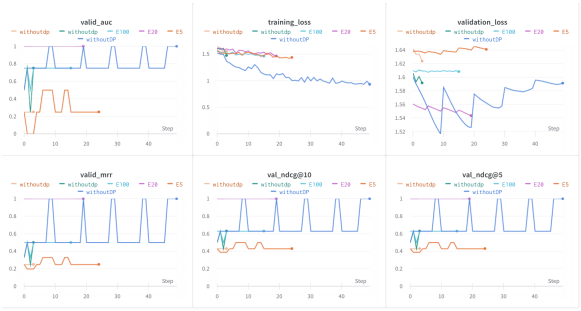
1) Need to curate sample of MIND Dataset from the original dataset
2) Need to see the variation of ϵ with the training loss.
1) Need to implement this via PyTorch Distributed on multiple clients(not simulation)

VII. Implementation via Torch Distributed

I have changed the code and made it compatible with the underlying structure of PyTorch Distributed. For the same I have created 5 ec2 t2-medium instances which have 2 core CPU and 4GB memory. 4 of theses instances will act as a client while training and the one left will act as a server which will aggregate all the parameters from the clients. Inbound rule for the clients allow any TCP request on random port 29603, and all outbound requests are allowed for a particular client. I have tried out many variants for the same that includes parameter averaging per epoch, parameter averaging after complete training, gradient averaging per epoch, gradient averaging after client's training. There is one other variant where server also have some data and takes part in the training and when server only does the aggregation. I have used slurm backend task scheduler that automatically runs the training script via PyTorch distributed as soon as the required resources are available. Since the PyTorch distributed package is only for distributed training, there are some issues that needs to be dealt with while dealing with federated learning.The results obtained are as follows:

1) When server also have some data and takes part in training with all the other clients, then PyTorch-distributed follows all-ring-reduce mechanism to distributed gradients/parameters to all other clients and the server, thus at the end of training all the clients and the server have the same model. The server can now distribute this model to all other new clients taking part in learning.
2) When server doesn't takes part in the distributed training there all clients will do the training on their own dataset and broadcast their parameters to the server for aggregation. But there are some issues that I faced while dealing with this scenario, because PyTorch Distributed package only allows you to send tensors to the other nodes taking part in distributed training. Some of them includes
   a) I am unable to send the client's model state dict() that contains the parameters of the model to the server, because it's an ordered dictionary not a tensor.
   b) If I send each tensor from client model parameters to the server separately then it becomes hard for the server's end to keep track of which client model parameter it is, because multiple clients will be sending their parameters at the same time.

   Some workaround for the same that I have used in my code is socket programming. The clients sends it model over TCP Connection to the server where server will average the weights and sends it back to the clients. The results and experiments obtained from the paper are as follows: The figure below shows the training statistics for a particular client.



However the issues that I faced from the above approach are the following:-

a) The proposed approach is not robust to client failure. If a client fails the entire training will fail.
b) The communication cost is very high. For each global epoch, each client model has to send 268Mb of clients data to the server.
c) Since the model has been trained from scratch, you can see a lot of irregularities in the graph of a node above. The client typically has very less amount of training data, so the metrics like NDGC, MRR are not evaluated over a larger dataset. Some other approaches to mitigate these solutions can be:
   i) We can use a more robust federated learning framework like flower PyTorch.
   ii) To avoid the irregularities in the graphs made we could first train a central model with the data and then distribute the model to the clients. This will help in faster convergence and less ups and down in the clients training graphs.
   iii) To reduce the communication cost, instead of sending the entire model, we should only send the layers that have been changing due to training.
   iv) To mitigate the problems of stragglers, we can employ various techniques given in the paper async SGD.

VIII. Issues in the paper

1) How user model at the client-end is being used. They have just reinitialized it at each training step. I am assuming that they are just distributing user model at the server to client but it is not clearly mentioned what they are doing
2) There is no clear way of generating news vector embeddings at the user's end while making a prediction. User model at the client requires news vectors but they have very vaugely created news vector at the server which is a breach of privacy.
3) Although they have mentioned that have reduced communication overhead between client and server, but there is no derivation of that in the entire paper of how they calculated it.
4) They have scaled training loss in code by a factor which is increasing, which is not very clear as to why they do it. This i think in incorrect.
5) I have retrained code over and over for addressa dataset and unable to replicate the result.
6) The way they are doing Multiparty computation in code is seriously violating privacy of user.

IX. Conclusion

[1] Tao Qi, Fangzhao Wu, Chuhan Wu, Yongfeng Huang, and Xing Xie. 2020. Privacy-preserving news recommendation model learning

[2] Defending Substitution-Based Profile Pollution Attacks on Sequential Recommenders by Zhenrui Yue, Huimin Zeng, Ziyi Kou, Lanyu Shang and Dong Wang

[3] FedRecAttack: Model Poisoning Attack to Federated Recommendation by Dazhong Rong, Shuai Ye, Ruoyan Zhao, HonNing Yuen, Jianhai Chen and Qinming He

[4] Ditto: Fair and Robust Federated Learning Through Personalization by Tian Li, Shengyuan Hu, Ahmad Beirami 2, Virginia Smith 1