Distributed Graph Convolutional Networks with Independent Subnet Training

Anonymous Authors[1]

Abstract

The overall goal of this project is to develop novel methodology for efficiently training graph convolutional networks (GCN). Recent researches towards efficient GCN mainly focus on node partition, which can be further divided into two branches of layer sampling (e.g. GraphSAGE (Hamilton et al., 2017), FastGCN (Chen et al., 2018), LADIES (Zou et al., 2019)) and graph sampling (e.g. ClusterGCN (Chiang et al., 2019), GraphSAINT (Zeng et al., 2019)), whereas another path of model parameter partition is understudied. Our exploration will fall into three steps: i) starting from parameter partition by making connection with independent subnet training (IST) (Yuan et al., 2019); ii) integrating the developed approach from parameter partition to node partition, trying to developing novel sampling/clustering methodologies; iii) building a systematic distributed training system to support the proposed method. This report mainly focus on the first step, leaving the others further work.

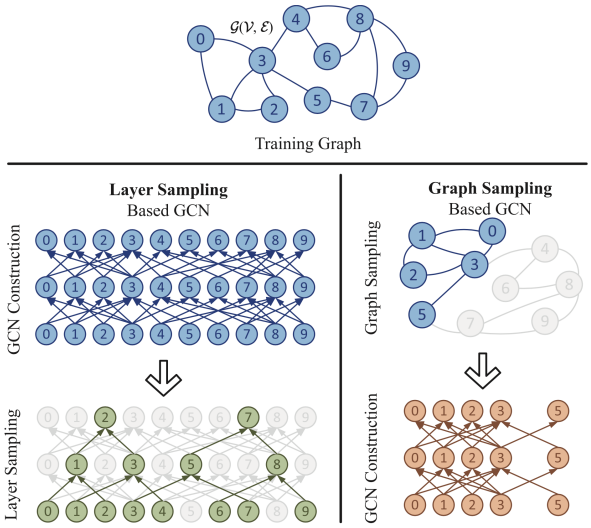

Figure 1. Two mainstream pathways towards efficient and scalable GCN. (Credit to (Zeng et al., 2020)). They both target on the problem of overwhelming number of nodes. However, in this paper, we focus on the alternative aspect in model parameter space

1. Introduction

Deep neural networks (DNNs) have been dominating a large quantities of tasks and even surpassed human-level performance in many fields such as natural language processing (Olsson, 2009; Schmidt & Wiegand, 2017) and computer vision (Liu et al., 2017; Ioannidou et al., 2017; Alom et al., 2018). One main component that contributing to DNN's significant achievement is that, the abundant real-world information such as voice and pictures are presented in a well-structured Euclidean space, where some important operations (e.g. convolutions) can be easily performed. However, Euclidean space is not applicable to all existing data, as a large number of application relies on the data in a graph structure. Representative applications include chemistry, where molecules are modeled as graphs, and more obviously, social networking where every user is regarded as node and edges exist when users have some social connections. To transfer the power of deep learning methods from Euclidean space towards non-Euclidean space, researchers attempted to developed several neural networks for graph data, and most classic and popular model is graph convolutional networks (Kipf & Welling, 2016), which generalizes the classic convolutional operation for graph scenario, yielding successful performance.

Although GCN is shown effective on various of tasks including both node-level and graph-level classification (Kipf & Welling, 2016), it is notorious on its disability of huge-graph training. In practical, GCN training requires to save the whole graph data and all hidden representation of all nodes into memory. Even when the graph is huge, GCN is still trained in full-batch manner, which is extremely vulnerable to memory overflow problem.

To deal with the aforementioned problem, recent researches

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

towards efficient GCN mainly focus on node partition, which can be further divided into two branches of layer sampling (e.g. GraphSAGE (Hamilton et al., 2017), FastGCN (Chen et al., 2018), LADIES (Zou et al., 2019)) and graph sampling (e.g. ClusterGCN (Chiang et al., 2019), GraphSAINT (Zeng et al., 2019)). The main idea of these individual path is depicted as Figure 1 and clarified in details in Section 2.

Notice that the unified goal of node partition methods (both layer and graph sampling) is, to seperate the huge graph into multiple small graphs for mini-batch training. In this case, GCN is not only trainable on large graph eventually, but also open to distributed training.

Different from the previous two paths of efficient GCN methods, which takes both memory costs and distributed training into consideration (generally, they pay more attention on the memory problem), in this paper, we emphasize on the distributed training for GCN, and more specifically, we target on the understudied efficient GCN method that trains the model in parallel by seperating/partitioning GCN model parameters into sub-models. Hence, the goal of our method is only for proposing a distributed method to accelerate GCN training in parallel. The method is originated from independent subnet training (IST) method (Yuan et al., 2019) and hope to be generalized into graph setting. Therefore, we name our method as GCN-IST, or GIST for short. The details of our methods will be introduced in Section 3.

Experimental results shows the feasibility of GCN-IST and can achieve linear speed-up on small graph dataset. Further work on deeper GIST with larger graph is awaiting.

Before we enter into the introduction of previous related works and our method, we would like to highlight few overall evaluation on this topic here.

Some excitement from this topic: Generally, to authors' knowledge, this work is the first attempt to generalize IST into GCN settings. It shows that GCN can be trained by multiple GPUs simultaneously, showing another way to accelerate GCN training other than operations on node space.

Some concerns about this topic: However, considering the initial motivation of IST work is that, current deep nerual networks, such as ResNet, can be as deep as thousand of layers, regarding the large model with a large number of parameters that requires large memory, IST can significantly alleviate the memory problem and also enable fast training on small submodels. Whereas most of GCN models are small and only have few hidden layers, the significance of IST on GCN is generally not as great as original IST.

The following part of the paper is seperated into five parts. We will first introduce the preliminary knowledge, including GCN, IST and existing methods for distributed GCN in Section 2. Our methods are then illustrated in details in Section 3 The variant of our method is also mentioned in the stated section. Section 4 shows the experiments results. The paper is ended with a brief conclusion in Section 5.

2. Preliminary

In this section, we first introduce the basics of GCN and then cover some existing methods for efficient GCN training.

2.1. Graph Convolutional Networks (GCN)

In this project, we focus on the Graph Convolutional Networks (GCN), the most standard model among all graph neural networks. The forward propagation rule for a GCN (Kipf & Welling, 2016) is defined as follows:

$$H_{t+1} = \sigma(\bar{A} H_t \theta_t) \tag{1}$$

where σ is an activation function (e.g., ReLU), θ_t is a linear projection matrix for a given layer t, $\bar{A}$ is the normalized adjacency matrix A of the graph with added self-connections as $\bar{A} = \hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}}$, where $\hat{A} = A + I_N$, and $\hat{D}$ is the diagonal degree matrix for $\hat{A}$. Finally, $H_0 = X$ where X is the input data.

Later on, many variants and modifications emerges, such as graph attention network (GAT) (Veličković et al., 2017), Mask-GCN (Yang et al., 2019), Confidence-based GCN (Vashishth et al., 2019). However, in this paper, we only focus on the classic and standard GCN.

2.2. Efficient GCN Benchmark

The bottleneck of current efficient GCN methods is majorly on the massive quantity of nodes. To solve this problem, recent researches towards efficient GCN can be divided into two branches of layer (node) sampling (GraphSAGE (Hamilton et al., 2017), FastGCN (Chen et al., 2018), LADIES (Zou et al., 2019)) and graph (node) sampling (ClusterGCN (Chiang et al., 2019), GraphSAINT (Zeng et al., 2019)), as shown in Figure 1.

For layer sampling branch:

GraphSAGE (Hamilton et al., 2017): Most machine learning techniques on graphs are transductive in nature: they require knowledge of the entire graph during training. GraphSAGE presents an inductive formulation that can be generalized to unseen graphs. GraphSAGE learns a set of *aggregator functions*, which iteratively aggregate information in a node's local neighborhood. Aggregation continues for K total steps. During each step, a node aggregates the representation of its neighbors, concatenates it with its own current representation, passes it through a linear layer, and applies a non-linear activation function. As nodes aggregate information from their local neighborhood iteratively, the receptive

field of the aggregation functions expands exponentially. In practice, GraphSAGE only samples a fixed-size neighborhood and only computes representations for nodes that fall within the receptive field (i.e., smaller portions of the graph can be considered at a time). The entire architecture can be trained with gradient descent, and multiple choices exist for the neighborhood aggregation function (e.g., max pooling, averaging, summing, LSTM, etc.).

FastGCN (Chen et al., 2018): The embedding function for each layer is defined as an integral over a probability distribution parameterized by the nodes and edges within the graph. This operation resembles a continuous representation of the GCN forward propagations rule and is shown below:

$$\tilde{h}^{(l+1)}(v) = \int \hat{A}(v,u)h^{(l)}(u)\theta^{(l)}\mathrm{d}P(u) \qquad (2)$$

To evaluate this integral in practice (i.e., compute the actual note representation), a Monte Carlo approach is adopted. In other words, they randomly sample a fixed-size group of nodes from the graph and use this to compute the representations of a given layer. Furthermore, an layer-wise importance sampling scheme, based on connectivity nodes, is adopted to favor sampling of nodes with high degree (i.e., this reduces the variance of the estimator). In comparison to GraphSAGE, this method limits the number of nodes considered to the sum of nodes sampled at each layer (i.e., the receptive field of GraphSAGE expands exponentially). In practice, this method is implemented by using a forward pass similar to the vanilla GCN, be only sampling a small group of nodes to be considered at each layer of the model. The probability distribution for such node sampling at each layer of the GCN is weighted by the degree of each node.

LADIES (Zou et al., 2019): LADIES aims to solve two problems: i) the exponentially expanding receptive field of GCN; ii) the sparse connectivity in layer-wise sampling (i.e., as proposed in FastGCN). In particular, because FastGCN samples nodes independently for each layer, this creates sparsity in connections between layers. Therefore, LADIES introduces a layer-conditional approach, where the nodes selected depend on those selected in the layer above. This importance sampling scheme is designed in a top down manner, where nodes are randomly sampled at the top layer, then each lower layer performs node sampling based on the nodes that have been sampled in the layer above. The importance of nodes at a given layer is defined by the number of connections is creates to active nodes in the layer above. In practice, this can be implemented by simply sampling nodes at each layer using this importance scheme, then performing a forward pass resembling the normal GCN forward pass. However, only active nodes that were sampled are considered.[1]

[1]It should be noted that for all node sampling schemes (e.g., LADIES, FastGCN), the Laplacian matrices are reconstructed and normalized after sampling occurs, which allows the neighboring representations to be scaled appropriately (i.e., this accounts for the presence of fewer active neighbors).

For graph sampling branch:

ClusterGCN (Chiang et al., 2019): Cluster-GCN uses graph clustering to sample a densely-connected subgraph within a global graph. This subgraph is then considered during the forward pass, while other nodes are not. Such node partitioning allows the GCN memory footprint to be minimized and also creates the possibility of developing deeper GCNs by restricting the receptive field. The subgraph, in this case, is considered our mini-batch. So, we no longer perform full gradient descent, but rather stochastic gradient descent to train this algorithm. Once subgraphs are created, the adjacency matrix is reconstructed into several sub-matrices (i.e., all connections that are not within a subgraph are zeroed out). The same procedure is followed to divide the node features and weights accordingly in a "block-diagonal" fashion. To construct subgraphs, the authors propose a procedure that maximizes connectivity within each subgraph. To combat issues with removing links or subgraphs being biased toward containing similar nodes, the authors modify the procedure to produce a very large number of clusters, and then randomly sample and union a group of these clusters to form each subgraph/mini-batch.

GraphSAINT (Zeng et al., 2019): Similar to ClusterGCN, GraphSAINT randomly samples a subgraph during each iteration of training for the GCN forward pass, instead of sampling a group of nodes at each layer of the GCN. Such a formulation eliminates issues with receptive field expansion. Deviating from ClusterGCN, GraphSAINT considers the bias created by unequal probabilities of sampling nodes during subgraph construction and proposes normalization techniques to eliminate this bias. Given some sampling method `sample`, they pre-process the graph to assign a sampling probability to each node or edge (i.e., sampling can be done based on node and/or edge probabilities). Many choices exist for `sample`, but it should generally encourage "similar" nodes to be within the same subgraph. A subgraph is then sampled independently based on this probability distribution over nodes. Then, training is conducted using SGD by processing each sub-graph as a mini-batch and updating model parameters accordingly. During the forward pass, each entry $A_{u,v}$ within the subgraph's adjacency matrix is scaled inversely by a normalization term $\alpha_{u,v} = \frac{p_{u,v}}{p_v}$ (i.e., the upper term is a node probability and the lower term is an edge probability). This normalization terms accounts for the bias created by our sampling scheme. Variance is minimized by developing samplers that are likely to sample similar nodes within the same subgraph (i.e., this intuitive idea is rigorously proven in the theory of the paper).

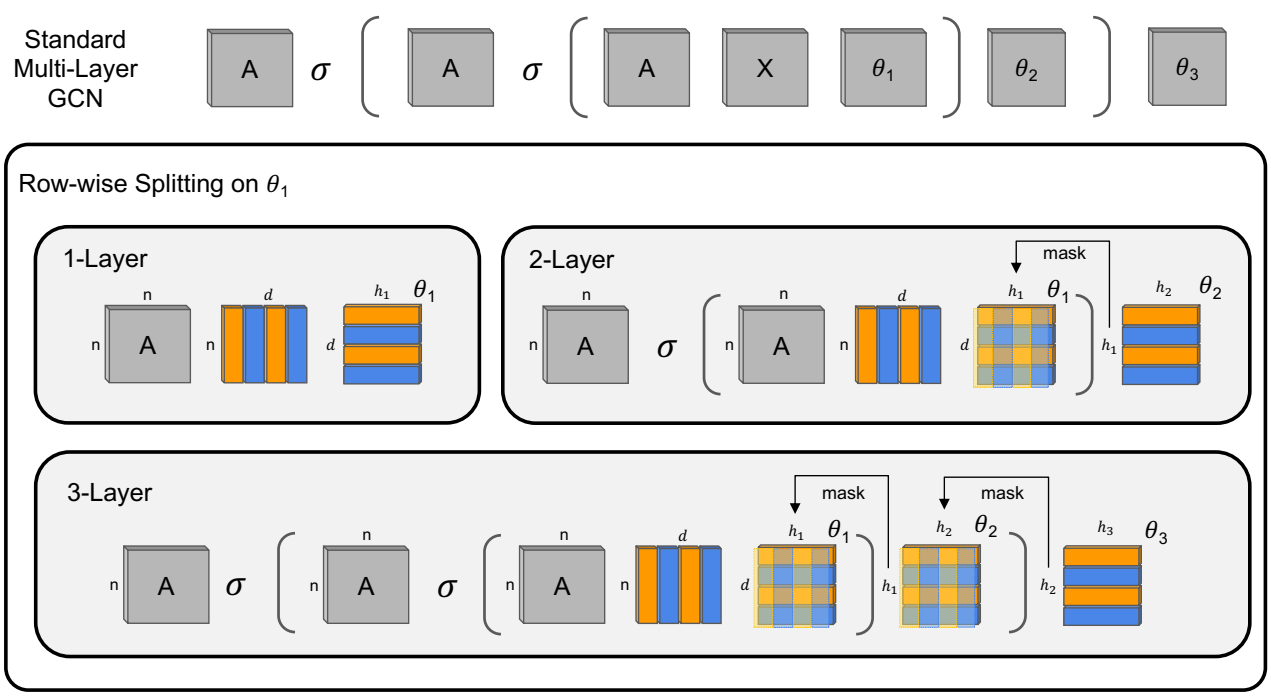

Figure 2. Diagram of the proposed GCN-IST method. The method can be applicable to GCN models with arbitrary number of layers. GCN-IST is featured by an active row-wise partitioning for any parameter θ starts from a row-wise partitioning for parameter θ_1, which performs as dividing node features into subnets.

The taxonomy is based on the phase when node partition is determined. Obviously, these two categories is not mutually exclusive, so some reasonable combination can be beneficial.

Apart from huge node size, the high dimension of node features is sometimes problematic. In this work, we also consider feature partition for efficient GCN training. Admittedly, the problem could be simply solved by traditional preprocessing ways such as dimensionality reduction and therefore drawing little attention from GNN community, we regard this setting as a simplest task to trigger our explorations.

2.3. Independent Subnet Training (IST)

Independent subnet training (IST) is a novel distributed training method for neural networks. By stochastically dividing the origin model, which is often large, into several non-overlapping subnets, IST trains these subnets simultaneously. The computation of every single subnet saves a lot of time and resources since a thinner model is allocated to each worker. In this case, IST significantly accelerates the training process comparing with standard data parallel approaches for distributed learning, and IST scales to large models that cannot be learned using standard data parallel approaches. In this paper, we take the idea of IST and generalize it to GCN setting.

3. GCN-IST: Applying IST to GCN

In this section, we will introduce the proposed method that divides the origin GCN method into submodels (so-called subnets) for IST training. The proposed method is depicted in Figure 2, following a variant that also worth considering. When encountering large graph, we rely on ClusterGCN for node partition. Within every subgraph, we continue our standard GCN-IST implementation. The collaboration between ClusterGCN and GCN-IST is explained in Section 3.3.

3.1. GCN-IST for Small Graph

Our proposed GCN-IST method can be clearly explained assisted by Figure 2. As our initial goal is to create submodels for GCN, there are two ways to do partitioning on GCN parameters. Namely, the θ matrix can be divided on the dimension of either row or column. In our work, we choose to actively split the weight matrix θ by row, leaving the counterpart the variant of our proposed GCN-IST method. As shown in Figure 2, if we only focus on 1st-layer splitting,

that is, only θ_1 is divided in row, in this case the input data X will be inevitably partitioned according to the splitting style of θ_1 (due to matrix multiplication rule). In the figure, parameter blocks with the same color will be put into identical subnet. Here, we find that, usually an active partition on one weight matrix will cause some other matrices divided accordingly. To rewind, for one-layer GCN-IST, when the parameter θ_1 is actively partitioned in row, the input matrix X is passively partitioned in column accordingly, acting as feature splitting (Every subnet only contains part of features dimensions).

Then, we come to 2-layer GCN-IST. Again, an active partition on row dimension is performed on both θ_1 and θ_2, correspondingly, apart from the feature splitting for input X, θ_1 is also separated column-wisely. Therefore, for θ_1 in Figure 2:2-Layer, only those intersection with pure color in θ_1 is allocated for subnets, $i.e.$, for θ_1 with 16 blocks, only those 4 pure-orange blocks are for orange subnet and those 4 pure-blue blocks are for blue subnet, leaving the rest 8 blocks unchanged during this iteration.

Every time the active splitting results are different so that with the stochastic partitioning and training process, we suppose the final model trained by GCN-IST will converge to that with standard but expensive training procedure.

3.2. A GCN-IST Variant

As mentioned in the last paragraph, there are two ways to do partitioning on GCN parameters. apart from actively splitting the weight matrix θ by row, there is another strategy that actively partitioning θ by column. In appendix, Figure 5 shows the variant of our GCN-IST method. Complementary to our active row-wise partitioning GCN-IST, this variant uses active column-wise partitioning for any parameter θ that also starts from a row-wise partitioning for parameter θ_1. Accordingly, other weight matrix in the next GCN layer will be passively partitioned in row. We will compare these two methods and find the superiority.

3.3. GCN-IST for Large Graph

For large graph, we use node partition method from standard ClusterGCN to divide the entire graph into small sub-graph. For every sub-graph, we use and explore our proposed GCN-IST. The pipeline is shown in Figure 3.3. However, the experimental result is still ongoing and not in this report.

3.4. Potential Directions

To further improve our method, a smart partitioning strategy is wanted. For example, the magnitude of weights/activations within the GCN could be used as an approximation of importance, allowing the partitioning to be conditioned on this importance. Two options exist for this: i) Utilize the magnitude of weights within H to inform the partition (i.e., importance sampling based on H); ii) Utilize the magnitude of weights within θ to inform the partitioning (i.e., importance sampling based on θ). Both of these methods could be tested empirically to determine which works better. The exploration is left for further work.

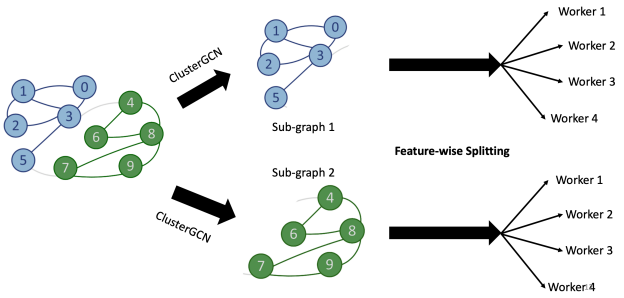

Figure 3. Pipeline for large graphs using CluterGCN with GIST

4. Experiments

In this section, after introducing the dataset information, we evaluate our GCN-IST by firstly on partitioning on first GCN-layer only and then promote to the following layers. Effects of local iteration is also explored.

4.1. Datasets

- Cora dataset (Sen et al., 2008) is a citation networks with 2708 nodes of machine learning paper. A standard stemming and stop word removal process is performed. Words with document frequency less than 10 are also removed and 1433 distinct words are finally kept, forming the node feature with 1433 dimensions. Edges exist between two paper whenever they share the same author and form 5429 links. 7 categories of machine learning paper are the label for the node classification task.

- Similar to Cora, CiteSeer dataset (Sen et al., 2008) has 3312 nodes of machine learning paper, 3703 distinct words in the vocabulary for node features and 4732 links in total for 6-class node classification task.

- Pubmed dataset (Sen et al., 2008) is another citation network with 19717 nodes, 500 distinct words and 44338 links for 3-class node classification task.

4.2. On Feature Splitting

Table 4.2 shows for more than 2 workers, layer norm will improve much the results, and IST in general can achieve similar best test accuracy (¡2%) with linear speedup.

Table 1. Experiments on GCN-IST with partitioning on first GCN layer only

IST?	#subnets	#Epoch	LR	LayerNorm?	Val%Last	Val%Best	Test%Last	Test%Best
False	1	400	0.1	False	82.00	84.67	82.9	83.8
True	2	200	0.025	False	82.33	84.67	83.8	84.6
True	4	100	0.05	True	80.67	83.33	77.6	82.1
True	8	50	0.05	True	78.00	84.67	77.6	81.8

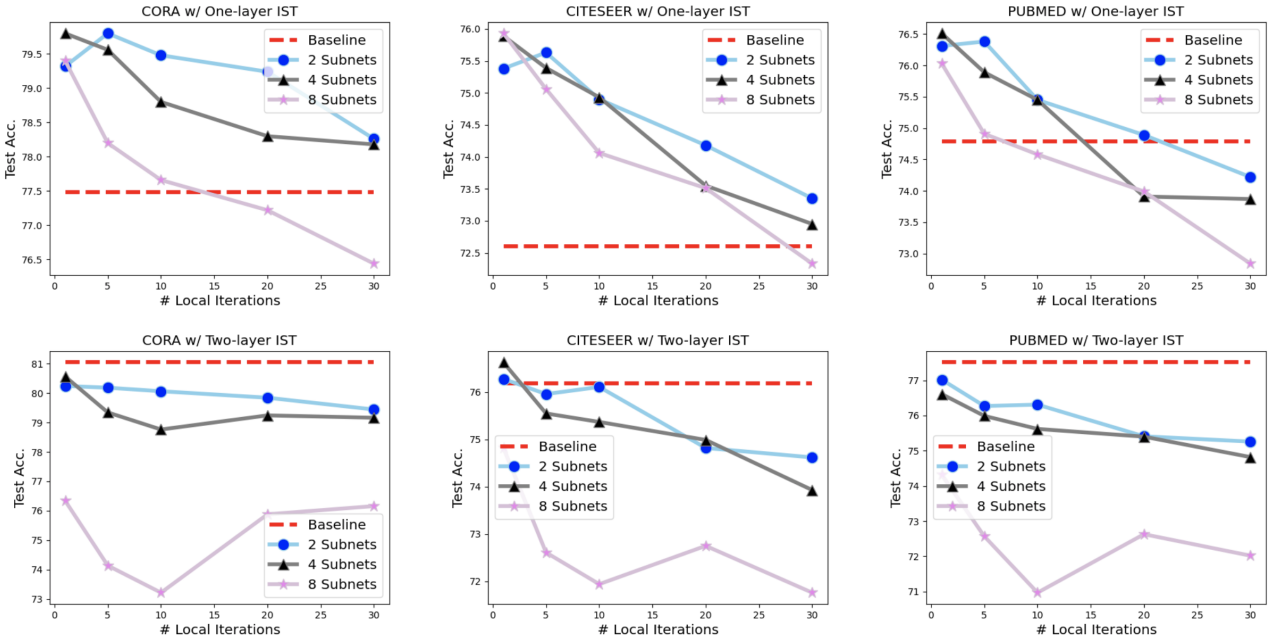

Figure 4. Results across different number of local iterations and subnetworks for all datasets. Results are included for both one layer and two layer IST. The hidden size for one layer and two layer IST were 16 and 64, respectively.

4.3. Incorporating Local Iterations

All results in Sec. 4.2 perform synchronization after every epoch. To reduce communication requirements, local iterations can be incorperated, allowing subnetworks to be trained for multiple epochs between synchronization rounds. Exact results over different number of local iterations are presented in 5 and visualized in Fig. 4, where are results are averages recorded across 5 trials. All tests use layer normalization and have a hidden dimension of 16.

4.4. Splitting Both GCN Layers

For the above experiments, only the first layer of the GCN was split. Although the first layer contains significantly more parameters than the second layer (i.e., 24K parameters vs. 150 parameters), splitting both layers of the GNN is an important experiment, as it proves IST is extensible to deeper architectures and architectures with larger hidden layers. Results are provided across multiple datasets in Table 5 (i.e., each metric represents an average across five trials). For these experiments, the hidden layer size was increased to 64, which was needed to achieve better performance with eight subnetworks (i.e., a hidden size of 16 would result in the hidden layer dimension becoming two for each subnetwork). It should also be noted that, when both layers are partitioned, layer normalization becomes essential to the convergence of the global model (i.e., otherwise the distribution of values within the hidden layer would be unstable, which becomes an issue when the output layer does not always see the entire hidden layer).

5. Conclusion and Further Work

This work focuses on distributed training efficient GCN using independent subnet training. Around half of the work is done and further work including i) ClusterGCN for large graph; ii) Comparison with GCN-IST variant; and iii) real distributed system is continuously undergoing.

References

Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Van Esesn, B. C., Awwal, A. A. S., and Asari, V. K. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

Chen, J., Ma, T., and Xiao, C. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. *arXiv e-prints*, art. arXiv:1801.10247, January 2018.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pp. 1024–1034, 2017.

Ioannidou, A., Chatzilari, E., Nikolopoulos, S., and Kompatsiaris, I. Deep learning advances in computer vision with 3d data: A survey. *ACM Computing Surveys (CSUR)*, 50 (2):1–38, 2017.

Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv e-prints*, art. arXiv:1609.02907, September 2016.

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., and Alsaadi, F. E. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

Olsson, F. A literature survey of active machine learning in the context of natural language processing. 2009.

Schmidt, A. and Wiegand, M. A survey on hate speech detection using natural language processing. In *Proceedings of the Fifth International workshop on natural language processing for social media*, pp. 1–10, 2017.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Vashishth, S., Yadav, P., Bhandari, M., and Talukdar, P. Confidence-based graph convolutional networks for semi-supervised learning. *International Conference on Artificial Intelligence and Statistics*, 2019.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Yang, L., Wu, F., Wang, Y., Gu, J., and Guo, Y. Masked graph convolutional network. In *IJCAI*, pp. 4070–4077, 2019.

Yuan, B., Kyrillidis, A., and Jermaine, C. M. Distributed Learning of Deep Neural Networks using Independent Subnet Training. *arXiv e-prints*, art. arXiv:1910.02120, October 2019.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. GraphSAINT: Graph Sampling Based Inductive Learning Method. *arXiv e-prints*, art. arXiv:1907.04931, July 2019.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. Accurate, efficient and scalable training of graph neural networks. *Journal of Parallel and Distributed Computing*, 2020.

Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., and Gu, Q. Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. *arXiv e-prints*, art. arXiv:1911.07323, November 2019.

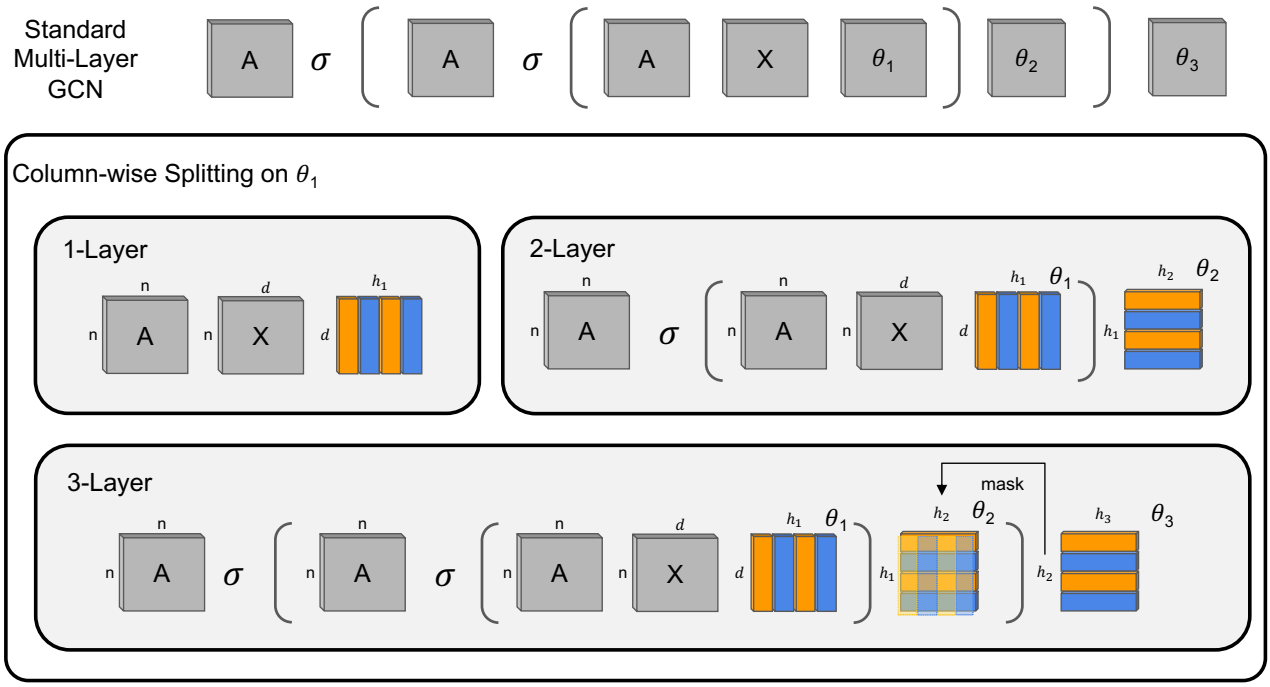

Figure 5. Diagram of the variant of our GCN-IST method. Complementary to our active row-wise partitioning GCN-IST, this variant uses active column-wise partitioning for any parameter θ that also starts from a row-wise partitioning for parameter θ_1.

Table 2. Full results for Table 1

Dataset	# local iteration	#subnets	LR	Test Best	Val Best	Test Last
CORA	1	2	0.05	79.32	78.84	76.26
	1	4	0.01	79.80	78.68	76.28
	1	8	0.05	79.40	77.88	75.64
	5	2	0.1	79.80	78.44	75.54
	5	4	0.05	79.56	77.48	77.28
	5	8	0.05	78.20	76.60	77.56
	10	2	0.1	79.48	78.44	77.70
	10	4	0.01	78.80	77.80	76.88
	10	8	0.005	77.66	76.84	77.44
	20	2	0.1	79.24	78.08	77.16
	20	4	0.1	78.30	77.32	76.92
	20	8	0.01	77.22	75.24	76.94
	30	2	0.05	78.26	77.40	76.32
	30	4	0.01	78.18	76.76	78.08
	30	8	0.01	76.44	74.68	76.44
CITESEER	1	2	0.05	75.38	75.26	69.32
	1	4	0.1	75.89	75.12	68.35
	1	8	0.05	75.93	75.14	71.85
	5	2	0.1	75.63	74.42	69.60
	5	4	0.1	75.39	75.06	72.20
	5	8	0.05	75.06	74.42	72.91
	10	2	0.1	74.90	74.26	72.35
	10	4	0.1	74.93	74.62	73.22
	10	8	0.005	74.06	73.90	72.78
	20	2	0.1	74.18	73.54	71.47
	20	4	0.1	73.55	72.96	72.86
	20	8	0.005	73.51	73.20	72.66
	30	2	0.05	73.35	73.36	71.84
	30	4	0.1	72.95	72.82	72.71
	30	8	0.005	72.33	72.00	72.33
PUBMED	1	2	0.05	76.31	76.56	70.95
	1	4	0.05	76.51	76.55	71.55
	1	8	0.05	76.03	75.92	72.05
	5	2	0.1	76.38	75.69	71.15
	5	4	0.1	75.89	75.88	72.98
	5	8	0.05	74.91	74.97	72.98
	10	2	0.1	75.45	75.12	72.89
	10	4	0.1	75.45	75.45	73.93
	10	8	0.005	74.58	74.44	73.56
	20	2	0.1	74.89	74.61	72.72
	20	4	0.05	73.91	73.45	73.24
	20	8	0.05	73.99	73.81	73.21
	30	2	0.05	74.22	74.33	72.93
	30	4	0.1	73.87	74.05	73.33
	30	8	0.005	72.84	72.91	72.75

Table 3. Full results for Figure 4

Dataset	# local iteration	#subnets	LR	Test Best	Val Best	Test Last
CORA	1	2	0.1	80.24	78.92	76.64
	1	4	0.01	80.56	79.24	78.34
	1	8	0.01	76.34	74.60	69.98
	5	2	0.005	80.18	79.04	76.50
	5	4	0.01	79.34	77.56	77.86
	5	8	0.005	74.14	72.00	72.86
	10	2	0.05	80.06	79.00	77.80
	10	4	0.01	78.76	76.80	77.86
	10	8	0.005	73.22	72.20	73.18
	20	2	0.005	79.84	78.44	77.02
	20	4	0.005	79.24	77.68	78.84
	20	8	0.005	75.88	74.76	75.26
	30	2	0.05	79.44	78.88	78.14
	30	4	0.005	79.16	77.92	78.12
	30	8	0.05	76.16	74.40	60.72
CITESEER	1	2	0.05	76.28	76.10	70.87
	1	4	0.01	76.63	75.70	73.16
	1	8	0.01	74.82	73.44	70.23
	5	2	0.1	75.96	75.42	71.15
	5	4	0.005	75.55	74.98	73.72
	5	8	0.005	72.61	71.50	71.74
	10	2	0.1	76.11	74.96	73.43
	10	4	0.01	75.37	74.16	73.31
	10	8	0.005	71.94	71.08	71.92
	20	2	0.01	74.82	73.64	70.90
	20	4	0.1	74.99	74.20	74.10
	20	8	0.005	72.75	71.98	72.43
	30	2	0.1	74.62	74.06	73.01
	30	4	0.05	73.93	73.36	72.68
	30	8	0.005	71.76	70.80	71.65
PUBMED	1	2	0.1	77.02	77.20	69.97
	1	4	0.05	76.61	76.67	73.75
	1	8	0.01	74.32	73.93	70.37
	5	2	0.1	76.27	76.20	72.60
	5	4	0.01	75.99	75.76	74.90
	5	8	0.01	72.57	72.05	70.75
	10	2	0.1	76.31	75.71	73.80
	10	4	0.01	75.62	75.03	73.73
	10	8	0.005	70.97	70.80	70.79
	20	2	0.01	75.41	75.00	72.08
	20	4	0.1	75.40	74.83	74.43
	20	8	0.005	72.63	72.55	71.93
	30	2	0.1	75.26	74.88	73.72
	30	4	0.1	74.82	74.72	71.06
	30	8	0.01	72.02	71.60	69.62

Dataset	# local iteration	#subnets	LR	LayerNorm?	Val%Best	Test%Last	Test%Best
CORA	1	2	0.05	True	79.40	76.64	80.64
	1	4	0.05	True	78.36	76.28	79.60
	1	8	0.05	True	72.72	60.48	73.70
	10	2	0.1	True	79.24	76.86	79.18
	10	4	0.01	True	75.04	75.60	76.38
	10	8	0.01	True	60.36	61.56	62.08
	20	2	0.05	True	78.32	77.78	78.90
	20	4	0.01	True	74.80	76.56	76.56
	20	8	0.01	True	54.40	57.00	57.00
CITESEER	1	2	0.1	True	75.64	70.72	71.02
	1	4	0.05	True	74.68	72.43	75.82
	1	8	0.01	True	71.28	66.80	72.47
	10	2	0.1	True	75.26	72.60	74.96
	10	4	0.01	True	72.94	72.05	73.10
	10	8	0.01	True	63.00	63.16	63.77
	20	2	0.1	True	73.88	72.80	74.14
	20	4	0.01	True	71.36	71.63	72.67
	20	8	0.01	True	57.44	58.67	58.67
PUBMED	1	2	0.05	True	76.51	72.54	76.57
	1	4	0.1	True	75.67	71.55	75.52
	1	8	0.05	True	72.96	65.15	73.03
	10	2	0.1	True	75.73	73.53	75.52
	10	4	0.01	True	73.83	73.14	74.05
	10	8	0.01	True	65.55	64.81	65.51
	20	2	0.1	True	74.88	73.61	74.83
	20	4	0.01	True	73.08	72.51	73.45
	20	8	0.01	True	61.12	59.41	62.12

Benchmarking Low-Rank Matrix Completion Algorithms

Joshua Engels [*1] **Richard Morse** [*2]

Abstract

Low-rank matrix completion is an important problem with many applications, but existing algorithms are tested on many different datasets in non standardized ways, making true empirical comparisons between algorithms all but impossible. The first part of this paper surveys the existing algorithms for low rank matrix completion. The second part describes a tool we built to benchmark matrix completion algorithms in a standardized way. Using the tool, we examine the performance of low rank matrix completion algorithms on both synthetic matrices parameterized by size, rank, condition number, and percent of the matrix shown and on matrices with real world data.

1. Introduction

Low-rank matrix estimation (1) is a fundamental problem in computer science and machine learning, with far reaching applications in almost any big-data problem setting.

$$\min_{M:rank(M)\leq r} f(M) \tag{1}$$

Low-rank matrix completion (2) is one of the most studied and influential sub-classes of this problem, with applications in recommendation systems, image recovering and compression, as well as compressed sensing (Candes & Plan, 2010). The efficient solution to this problem is important to practitioners in a wide variety of fields, as, for example in 2009, the Netflix contest awarded a prize of $1 million dollars to the team which most accurately solved their matrix completion problem, delivering movie recommendations to customers based on limited preference data.

In many applications, the problem can quickly grow massive in size, with millions of individual entries in the underlying "ground truth" matrix. This burden makes exact solutions of the problem often computationally intractable. Hence the study of matrix completion turns from exact recovery to efficient solutions which approximate the ground truth matrix well, where the success of the approximation is defined by the problem setting. Moreover, in the general setting of full rank matrices, matrix completion is impossible; luckily, in the real world most ground truth matrices are low rank. Hence, the matrices have limited independent measurements and can be recovered using bounded space and time. The problem then changes into the form of finding a low-rank matrix, which is as close to the ground truth matrix as possible, in the space of the measured entries. The space of the measured entries is defined by the masking operator P_Ω, which permits that some fraction (generally small) of the ground truth matrix is observed. This problem is formulated as (2).

$$\min_{M:rank(M)\leq r} \frac{1}{2}||P_\Omega(M - M^*)||_F^2 \tag{2}$$

There exist many different algorithmic methods of approximately solving the matrix completion problem. Many algorithms such as (accelerated) factored gradient descent (Zhou et al., 2020), alternating (steepest) descent (Tanner & Wei, 2016), and power factorization, factorize the original matrix into smaller matrices, and solve the minimization problem on the factors. These methods have the advantage of incorporating the low-rank and positive semi-definite constraints of the original problem inherently into the new problem, and simplify the overall computations, but can require careful bookkeeping to keep the problem from diverging. Other algorithmic techniques such as Reimann methods (Mishra & Sepulchre, 2013), and iterative hard thresholding (Kyrillidis & Cevher, 2014) (Tanner & Wei, 2013), project the solution iterates onto a particular low-rank subspace to maintain the low-rank constraint, but work in the size and scope of the original matrix.

With proper initialization and careful selection of hyperparameters, these methods can perform well in various settings, however direct comparisons between methods can prove difficult. Authors can provide theoretical bounds for the performance of their algorithm under specific conditions, however, the important question still remains of how these algorithms perform in the real world.

To help answer this question, authors will often implement

[*]Equal contribution [1]Computer Science Department, Rice University, Houston, Texas, United States [2]Computational and Applied Mathematics Department, Rice University, Houston, Texas, United States. Correspondence to: Joshua Engels <jae4@rice.edu>, Richard Morse <rmm13@rice.edu>.

their algorithm using informed techniques, such as careful initialisation and selection of hyper-parameters, for the problem at hand. In general, these implementations are quite effective, and allow the new algorithm or technique to demonstrate its potential for real world adoption. However, the empirical testing and comparison of these algorithms is seriously lacking. Implementations are often only tested on a small number of specific datasets, and only compared to a small number of specific algorithms, usually those closely related to the one of interest. As different papers compare different algorithms on different datasets, it is hard to know how the overall landscape of algorithms compare, in any sort of objective sense. This seriously hinders practitioners and readers alike. Algorithm designers cannot discern how their algorithm or its implementation compare in the broad scope of the problem, and have a difficult time objectively verifying their results. Meanwhile, readers might seriously doubt the reproduciblility or impartially of an algorithms testing, given the limited scope of any individual papers empirical analysis. If an interested reader were looking for an algorithm to solve their particular problem, it would be quite difficult to make an informed decision of which one to use.

In this paper, we aim to remedy the issue by proposing a benchmarking tool for the low-rank matrix completion problem. Our MATLAB framework provides a standardized way for algorithmic implementations to be benchmarked and compared in various settings. The framework has already been used to compare several algorithms, and is designed for new algorithms to be easily integrated into it. The framework is built so that algorithms can be benchmarked using various comparison metrics, however the popular metric of time-constrained error was used in our analyses.

Such an idea is not entirely new: we were inspired by other comparison tools in the literature, such as the ANN-benchmarking tool for approximate nearest-neighbour search (Aumüller et al., 2020). The need for these tools is clear, and we hope that our framework can help bridge the gap in the low-rank matrix estimation setting.

In section 2 we overview some of the current literature in efficiently solving the low-rank matrix completion problem, explaining the popular algorithms which we test in our framework. In section 3, we introduce our tool and its functionality. In section 4, we show some empirical results demonstrating the use of our framework. In section 5 we conclude and provide points for future work, where the functionality and scope of our framework can be expanded to broaden its reach.

Please note that while we hope to empirically test algorithms we are in fact only testing implementations of algorithms. To properly compare algorithmic concepts, testing of multiple distinct implementations of an algorithm might be necessary; extensively optimizing just a single algorithm to perform well in benchmarking is a research project of its own. Despite this limitation, by testing multiple algorithms (directly from their authors) which use the same algorithmic approach, such as factorization-based or projection-based approaches, we can robustly compare these classes of algorithms, and illuminate their relative advantages and disadvantages of these larger algorithmic classes.

2. Existing Algorithms

We first introduce and examine the current state of low rank matrix completion algorithms.

2.1. Factorization Based Approaches

2.1.1. Factored Gradient Descent

The first algorithm we explore is a sort of "old faithful" of factored matrix completion approaches, factored gradient descent.

- Factored Gradient Descent
  The general idea behind factored gradient descent (FGD), like all factorization based approaches, is to break down the original problem into a smaller, simpler problem by factorizing the solution matrix and minimizing over the factor(s) as opposed to the original matrix. In FGD, we factorize the matrix to be minimized, $M \in R^{m\times m}$, as $M = UU^T : U \in R^{m\times r}$, and then substitute (2) with the following minimization problem.

$$\min_{U\in R^{m\times r}} ||P_\Omega(M^* - UU^T)||_F^2 \tag{3}$$

  Note that by the nature of using only one factor, FGD only works on square matrices. By factorizing the input problem as such, any low rank or positive semi-definite (PSD) constraint on M is intrinsically satisfied. For each step we move in the direction of the gradient of (3) w.r.t. the factor U, which simplifies to scaling the gradient of (2) by U.

- Accelerated Factored Gradient Descent
  Based off factored gradient descent, this method attempts to apply Nesterov Acceleration to FGD. This cannot be easily done because Nesterov Accelerated Gradient Descent (NAG), requires strong convexity in the objective to achieve an accelerated linear rate of convergence, but in factorizing the objective, we don't even have convexity guaranteed in the new problem. To overcome this hurdle, accelerated factored gradient descent (Zhou et al., 2020) proposes a variation of FGD, where Nesterov acceleration is applied in each step, and then the solution iterate is projected

onto a special set where restricted strong convexity holds. In this way, the algorithm is able to maintain the factorization of FGD along with the guarantees of NAG. The special set takes the form of $\Omega(U_0) = \{U \in R^{m\times r} : U^T U_0 \succeq 0\}$, which is the set of all matrices that are positive semi-definite when multiplied into the initial matrix U_0. With careful initialization of U_0, this algorithm achieves impressive theoretical guarantees, however its empirical comparison to a broader class of algorithms needs to determined.

2.1.2. Alternating Minimization

While similar in concept to factored gradient descent, alternating minimization approaches use two factors instead of one to decompose the original problem.

$$\min_{X\in R^{m\times r}, Y\in R^{r\times n}} ||P_\Omega(M^* - XY)||_F^2 \tag{4}$$

This allows alternating approaches to apply more generally to rectangular matrices and to be more robust in the types of problems they can solve.

- (Scaled) Alternating Steepest Descent
  The first alternating approach we examine is alternating steepest descent (ASD) as well as its slight variant, scaled alternating steepest descent (ScaledASD) (Tanner & Wei, 2016). The base of this algorithm is the simple alternating descent approach, where one factor is updated, and then the updated factor is used to take a step along the other factor, meaning factor updates occur in *alternating* succession. The updates for each factor are done via gradient descent with gradients computed as:

  $$\nabla f_X(Y) = -X^T(P_\Omega(M^*) - P_\Omega(XY))$$
  $$\nabla f_Y(X) = -(P_\Omega(M^*) - P_\Omega(XY))Y^T$$

  Alternating steepest descent veers from alterning descent in that it takes the steepest step at each iteration as opposed to one defined by a pre-determined step-size. The steepest step is solved for using basic calculus, and is found to be:

  $$t_x = \frac{||\nabla f_Y(X)||_F^2}{||P_\Omega(\nabla f_Y(X)Y)||_F^2}, t_y = \frac{||\nabla f_X(Y)||_F^2}{||P_\Omega(X\nabla f_X(Y))||_F^2},$$

  In the scaled variant of this algorithm, each gradient is scaled by an inverse factor as follows

  $$d_x = -\nabla f_Y(X)(YY^T)^{-1}$$
  $$d_y = (X^T X)^{-1}\nabla f_X(Y)$$

  This scaled variant is a quasi-newton method where the scaling factor serves as approximation of the objective's second order information and works to accelerate the algorithm, in theory.

- Low-rank Matrix Fitting (LMaFit) Low-rank matrix fitting (Wen et al., 2012) is an alternating method that can be seen as a relaxation of traditional alternating descent. Rather than take the ground truth matrix as a given, LMaFit parameterizes this input as a variable, and allows it to be optimized over throughout the problem (5).

  $$\min_{X,Y,M} ||P_\Omega(M - XY)||_F^2 : P_\Omega(M) = P_\Omega(M^*) \tag{5}$$

  This allows for easier closed form solutions to the subproblems within each alternating step, with limited to no loss in solution accuracy. This algorithmic idea forms the basis of LMaFit, however the algorithm also comes with another twist. Each step of LMaFit is weighted using the last iterate of the algorithm in a form of alternating acceleration. However, this acceleration is done quite carefully, with the weights initialised to 0, increased slowly when the algorithm stagnates, and then reset to 0 if the algorithm regresses. LMaFit performs quite well on traditionally difficult problems, where exact (or near exact) solutions are untenable. Taking advantage of its low per-iteration complexity, LMaFit is able to converge reasonably quickly, even when effective algorithmic descent proves difficult.

2.2. Projection Based Approaches

2.2.1. Iterative Hard Threshholding

Iterative Hard Thresholding is a proection based approach that is popular in a wide variety of matrix problems. The general idea is to use projected gradient descent, where vanilla gradient descent is augmented through a projection onto a low-rank subspace in each step. In this way, the solution iterates are kept low-rank, and "convex" guarantees are maintained through the powerful (rank) restricted isometry property (RIP) on the low-rank subspace.

$$(1 - \lambda(A))||X||_F^2 \leq ||A(X)||_2^2 \leq (1 + \lambda(A))||X||_F^2$$

where A is a linear operator, which would be the masking operator in the general matrix completion setting.
The exact subspace and projection step can vary between specific algorithms, but in the vanilla case the idea is to project onto a space of the top r singular vectors of the solution iterate, to maintain the rank-r constraint on the solution.

- Normalized Iterative Hard Thresholding
  The distinction between normalized and vanilla iterative hard thresholding is the selection of the step size at each iteration (Tanner & Wei, 2013). While the typical step size selection may be some constant or decreasing

function of the iteration number, in NIHT, the step size is carefully chosen as a function of the current solution iterate and its projection operator:

$$u_j = \frac{||P_U^j A^*(b - A(X^j))||_F^2}{||A(P_U^j A^*(b - A(X^j)))||_2^2}$$

where A^* is the adjoint operator of the sensing matrix, which is a surrogate for the masking operator.

- Conjugate Gradient Iterative Hard Thresholding
  Conjugate gradient IHT veers from the standard implementation, not in its selection of step size, but in its computation of the gradient at each step. As opposed to exactly computing the gradient in each iteration, which can be an exeedingly costly operation as the size of the problem grows, CGIHT pulls from the idea of conjugate gradient descent, computing a close approximation of the gradient. The close approximation is done using conjugate directions (Hager & Zhang, 2005) which are easy to compute and work quite well in maintaining descent throughout the algorithm. Hence, while each individual update step might not be as good, the time saved in computing the gradient allows more steps to be taken, and if the approximation is good enough this can improve the overall computation. This algorithm can work as an extension of iterative hard thresholding to massive matrix problems, where finding exact gradients and "best" update steps can be computationally prohibitive.

2.2.2. Riemann Methods

The class of Riemann methods all pull from the general idea of projecting the original matrix completion problem onto a Riemann manifold, which provides a new space and metric to optimize over.

- R3MC
  The general idea behind R3MC is to project the original problem onto the quotient space, which is defined by the equivalence class: $X = URV^T = (UO_1)O_1^T RO_2(VO_2)$, where O_1 and O_2 are two orthogonal r x r matrices. This equivalence class is augmented with a metric for the new space, one which is induced by the block approximation of the problem, where distances are now computed using the outer product of the inner R matrices. With equivalence class and metric in hand, the Rieman manifold has been defined, and the transformation of the problem onto this manifold reduces the problem space by a factor of r^2. The low-rank minimization problem is then solved on the Reiman manifold, and the solution is projected back to the original space, which is a trivial operation.

2.2.3. Least-Squares Methods

- MatrixIRLS
  MatrixIRLS, while not as popular as some of the above algorithms, was examined due to its effectiveness in solving *statistically hard* problems, i.e. matrix completion problems with only a very small fraction of entries shown, as well as matrices with very high condition number.

  MatrixIRLS is an iteratively reweighted least squares (IRLS) algorithm, with the addition of second-order Newton smoothing to escape local saddle points. The trick of this algorithm is that it is applied to the non-convex log-det objective, $\sum_i = 1^d \sigma_i(X + \epsilon I)$, where $\epsilon > 0$, is the smoothness parameter of the objective. In contrast to the original problem (1), this objective improves the scalability of the algorithm to commonly intractable problems, while giving up the convexity and smoothness of the original objective function. The non-convexity (and non-smoothness) of the new objective is dealt with through carefully smoothed steps and gradient updates. The algorithm proceeds by using the conjugate gradient method to solve a weighted least squares problem, with weights determined through the singular value decomposition of the solution iterate. The update step is then smoothed and the weights are updated for the next iteration. By improving the problem defining weights in this manner, the quadratic upper bounds on the log-det objective are improved at each iteration, providing a descent in the objective, even if no direct descent of the solution iterate is available. Furthermore, the smoothness parameter, ϵ, of the objective is maintained and updated throughout the algorithm, ensuring smooth convergence guarantees.

3. Benchmarking Tool

If one thing is clear from this literature review, it is that there are *many* different algorithms for low rank matrix completion. As we hinted at above, however, we found no real centralized source for algorithm comparison.

We remedied this problem by building a *centralised open source benchmarking tool*. The repository is hosted at `https://github.com/Matrix-Benchmarks/Low-Rank-Completion` and the website itself is hosted at `https://matrix-benchmarks.github.io/Low-Rank-Completion/`.

3.1. Tool Overview

The design of our benchmarking system is relatively simple, as it remains mostly a proof of concept. Building off the framework of (Kümmerle & Verdun, 2020), we designed a system to run matrix completion algorithms through a

variety of test matrices. The current algorithm testing framework itself is contained within MATLAB. Each algorithm is implemented in MATLAB as well, with each implementation based off author specifications and source code where available. We added instrumentation into each algorithm to integrate it into our framework, and give it the capability to be tested in the various ways we desire.

We made a decision early on to only care about one aspect of performance: how well the algorithm converges to the real matrix over time. In other words, as an experimental project we do not care about number of iterations to convergence or any other metric besides raw efficiency of the algorithm; otherwise, it is extremely difficult to compare completely different types of algorithms. One of the nice things about this idea is that in the future, individual algorithms within the framework can be tuned, so the performance of each algorithm represents the absolute best that the algorithm can do on the test matrices, and the best algorithm on a given matrix is the clear choice for a practitioner given a matrix of that type.

We strove to test every algorithm we discussed in our first section. Unfortunately, we were unable to get the accelerated factored gradient descent algorithm working. It seems that perhaps the algorithm only works on some specific initializations, or we just had something slightly wrong in our setup. Additionally, we are in the process of testing Matrix ALPS (Kyrillidis & Cevher, 2012). While Matrix ALPS shows promise so far, unfortunately we were not able to finish testing in time for this deadline (see our Github for current status). Other than these caveats, we met out goal of testing every algorithm listed above.

3.2. Test Data

The test matrices fall into 2 categories: real data and synthetic data.

- Real data: for now we have 2 types of real data we test with: image completion problems, where the algorithms get random pixels from a png image and guesses the others; and recommendation problems, where the algorithms get part of a user rating matrix and guess what the unknown ratings are. Specifically, we test with lena.png and the Movielens 1M dataset (Harper & Konstan, 2015), both standards in their respective fields.
- Synthetic data: we test the algorithms with random square matrices with all combinations of the following:
  - Size: 1000, 10000
  - Rank: 2, 20, 200
  - Percent of matrix "visible": 5, 10, 20, 40
  - Condition number: 2, 200, 20000

3.3. Running Experiments

Across both real and synthetic data, there are a total of 74 different parameter combinations. We skip 9 of these because the parameters of the matrix are below the information theoretic limit for low rank matrix recovery (Riegler et al., 2016), leading to a total of 65 different test matrices. We run each of the 11 algorithms described in section 2 for 1 minute on each matrix; thus a complete run to generate all of the performance data takes at minimum 12 hours. However, we also need to find the distance between each iteration of each algorithm and the goal matrix, which depending on the number of iterations and the size of the matrix can take just as long (e.g. finding the Frobenius distance between 1000 iterations of size 1000 matrices with a goal matrix takes on the order of a billion subtractions).

For each algorithm and matrix, we print out a time and distance pair to a file for each iteration. We then generate plots using Python's matplotlib that compare the performance of each algorithm on every matrix. Finally, we save all of these images as pngs in our github repository and then create a statically hosted github website from the repository allowing anyone to view the results. Note also that anyone can easily fork the repo and verify our results, or like we stated earlier try to improve the performance of an algorithm on a specific matrix (or include their own algorithm for comparison).

4. Benchmarking Results

We found that different algorithms perform better on different matrices; there is no single algorithm that is the universal best choice. Because we tested on too many matrices to individually examine here, we restrict our analysis below to a few "interesting" matrices and parameter regions, where "interesting" means examples where we can derive some useful observation about comparative algorithmic performance (matrices where every algorithm failed to make any progress at all or every algorithm quickly succeeded are usually much less "interesting", although they are still available on our website and we do analyze one in Figure 3).

We first will examine performance on synthetic matrices. For example, in Figure 1 we show an example from a region of hyperparemeters where MatrixIRLS dominates. We found that for small matrices (size = 1000), MatrixIRLS was able to converge to the real matrix on almost every example. It did so faster than other algorithms when the condition number was high, and in some cases like in Figure 1 it was the only one to converge at all. On larger matrices, on the other hand, MatrixIRLS sometimes was only able to run for a couple of iterations before the 1 minute timer ran out, and so performed the *worst* out of all algorithms (see Figure 2). This MatrixIRLS case study shows clearly how algorithmic performance can differ across the parameter space.

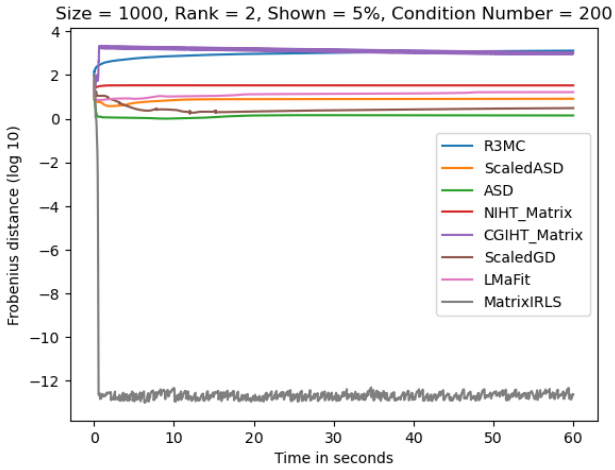

Figure 1. Benchmark of performance on a high condition number low rank matrix (size 1000, rank 2, the algorithm "sees" 5% of the matrix, condition number 200). MatrixIRLS clearly outperforms every other algorithm in these conditions.

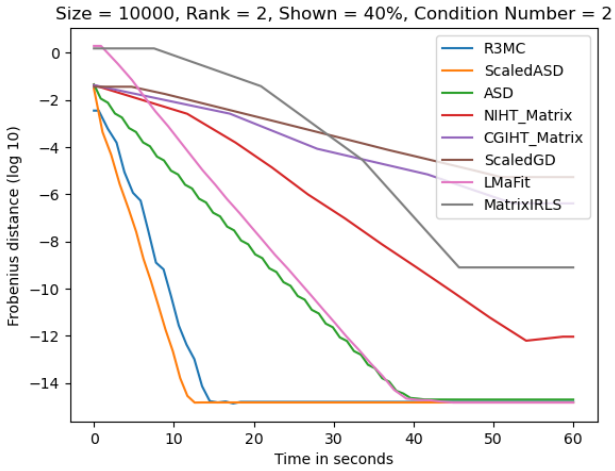

Figure 2. Benchmark of performance on a low condition number low rank matrix (size 10000, rank 2, the algorithm "sees" 40% of the matrix, condition number 2). Alternating descent and Riemanian methods perform the best.

Another interesting observation is the sharp frontier between synthetic matrices that are "solvable" and matrices where the algorithms make almost no progress at all. If we look at Figure 2, we see that nearly every algorithm converges almost exactly to the hidden matrix, and the ones that do not would probably converge in the next few minutes of running the algorithms. Contrast that to Figure 3, where almost no algorithm makes any progress towards the goal at all.

In fact, in Table 1, we see that almost every paramaterized

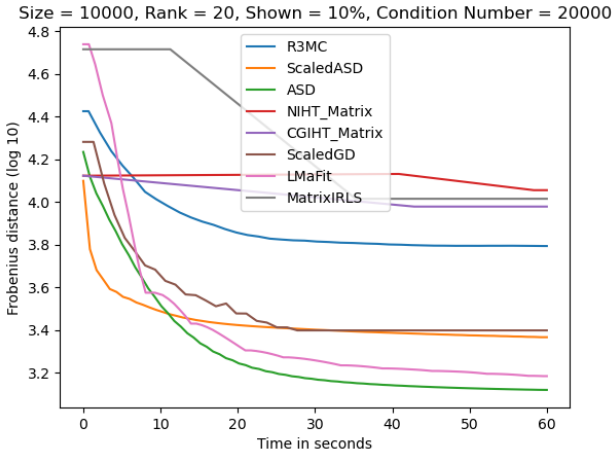

Figure 3. Benchmark of performance on a high condition number medium-low rank matrix (size 10000, rank 20, the algorithm "sees" 10% of the matrix, condition number 20000). No algorithm makes any real progress.

synthetic matrix either has an algorithm converge within a Frobenius distance of 10^{-10} or has no algorithm converge within even a Frobenius distance of 10^{0}. This result implies that there might be fundamental limits to our current low rank matrix completion methods; some matrices seem to be easily solvable, but others remain intractable, and there is very little in between. The matrices we observed all of our algorithms to struggle with were large matrices (size = 10000) with medium-high rank and medium-high condition number.

Table 1. Let d be the log base 10 of the closest distance any algorithm got on a matrix M. This table reports counts and percents of how many of our 63 synthetic testing matrices had d within a certain range.

LOG 10 FROBENIUS DISTANCE d	COUNT	PERCENT
$-17 < d < -10$	36	$\approx 57\%$
$-10 \leq d \leq 0$	7	$\approx 11\%$
$0 < d < 4$	20	$\approx 32\%$

Finally, the last result we analyze here is the performance of the algorithms on real datasets. For now, on real datasets our algorithm looks at how closely the matrix recovery algorthims can *predict* the unknown entries. This approach differs from the synthetic matrix testing procedure (where we find the Frobenius norm of the difference between the goal matrix and the algorithm's predicted matrix) because we *do not always have the entire goal matrix* in the real data case. For example, take the MovieLens dataset (algorithm results shown in Figure 4). The dataset is simply a set of (user, movie, rating) tuples, and not all users rate all movies.

We instead recreate a full matrix of size (number of users $\times$ number of movies), where each cell is filled in as the rating that a user gives to that movie. Then we give 30% of the entries of that matrix that we know to the algorithm (this is much smaller than 30% of the entire imagined matrix), and compare how well it predicts the other 70%. Note too that here we truncate the number of users from 6000 to 4000 to create a square matrix (there are 4000 movies); several of our tested algorithms do not work on rectangular matrices.

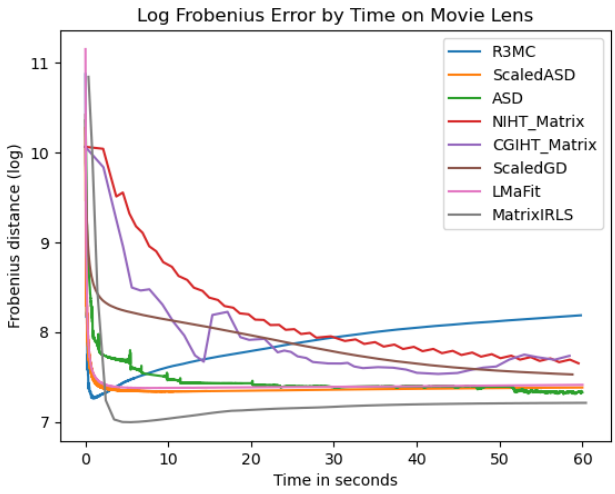

Figure 4. Benchmark of performance on the MovieLens 1M dataset. The algorithms are shown 30% of the known entries (around 2% of the entire matrix), and asked to predict the other known entries (error is the Frobenius distance of the predictions).

There are a few interesting aspects to Figure 4. The first is that all of the algorithms reach about the same minimum convergence distance. Indeed, we saw similar results for our main other real data application, image completion. Data in the real world seems fundamentally different from our synthetic data, though it is not immediately clear how we can remedy this problem so that our synthetic data is similarly difficult.

The second is that many of the algorithms actually get *worse* over time. We hypothesize this is because the actual matrix is not actually a low rank matrix, but that it is just approximated by one. Peoples' preferences are not *actually* determined by exactly 20 factors, so the algorithms overfit to the observed entries. Both observations point to the importance of testing with real world data, and perhaps developing new methods that can handle noisy and non-perfect low rank data.

5. Conclusions & Future Work

We have presented a tool that potentially offers an open source centralized repository for benchmarking matrix completion algorithms. If adopted widely, it will revolutionize low rank matrix completion, allowing students, practitioners, and researchers easy access to cutting edge matrix completion benchmarks.

There are a number of improvements we would like to make to our tool to speed it towards wide adoption.

- Expand the problem domain past matrix completion to the wider domain of matrix recovery. Not all of the algorithms we are testing right now will work in that wider domain, but we could have categories on the website for matrix recovery and for the specific category of matrix completion.
- Take the framework out of MATLAB. Right now the framework is heavily integrated with the algorithms themselves; we rely on the algorithms to correctly report intermediate results and timing. Ideally each algorithm could be written in any language, not just MATLAB, and would have an API allowing the benchmarking framework to pass in a matrix. The framework could then cut off the program after a minute. This change would lead to better separation between the framework and the algorithms (not to mention not having to deal with MATLAB's quirks). A more modular framework means that people can more easily improve and understand individual algorithms, as well as implement their own algorithms.
- Add more algorithms and datasets. For synthetic datasets, we want to test larger matrices and matrices with noise, as we hypothesize these will represent real world problems better. For real datasets, we want to expand the number of standard datasets we test with. Finally, we would like to add any other important algorithms from the literature or the matrix completion community, because this tool's usefulness directly relies on its completeness.

Ultimately, we believe that our tool can *already* provide interesting insights into how different matrix completion algorithms compare.

Acknowledgements

We would like to thank Dr. Kümmerle and Dr. Verdun for allowing us to build off of the MatrixIRLS testing framework, as well as each of the authors of the various matrix completion methods for making their code available to us. We would also like to thank Professor Kyrillidis for his support and guidance throughout this project.

References

Aumüller, M., Bernhardsson, E., and Faithfull, A. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020. ISSN 0306-4379. doi: https://doi.org/10.1016/j.is.2019.02.006. URL http://www.sciencedirect.com/science/article/pii/S0306437918303685.

Candes, E. J. and Plan, Y. Matrix completion with noise. *Proceedings of the IEEE*, 98(6):925–936, 2010. doi: 10.1109/JPROC.2009.2035722.

Hager, W. W. and Zhang, H. A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM Journal on optimization*, 16(1):170–192, 2005.

Harper, F. M. and Konstan, J. A. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015. ISSN 2160-6455. doi: 10.1145/2827872. URL https://doi.org/10.1145/2827872.

Kyrillidis, A. and Cevher, V. Matrix alps: Accelerated low rank and sparse matrix reconstruction, 2012.

Kyrillidis, A. and Cevher, V. Matrix recipes for hard thresholding methods. *Journal of mathematical imaging and vision*, 48(2):235–265, 2014.

Kümmerle, C. and Verdun, C. M. Escaping saddle points in ill-conditioned matrix completion with a scalable second order method, 2020.

Mishra, B. and Sepulchre, R. R3mc: A riemannian three-factor algorithm for low-rank matrix completion. *Proceedings of the IEEE Conference on Decision and Control*, 2015, 06 2013. doi: 10.1109/CDC.2014.7039534.

Riegler, E., Stotz, D., and Bölcskei, H. Information-theoretic limits of matrix completion, 2016.

Tanner, J. and Wei, K. Normalized iterative hard thresholding for matrix completion. *SIAM Journal on Scientific Computing*, 35(5):S104–S125, 2013.

Tanner, J. and Wei, K. Low rank matrix completion by alternating steepest descent methods. *Applied and Computational Harmonic Analysis*, 40(2):417 – 429, 2016. ISSN 1063-5203. doi: https://doi.org/10.1016/j.acha.2015.08.003. URL http://www.sciencedirect.com/science/article/pii/S1063520315001062.

Wen, Z., Yin, W., and Zhang, Y. Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm. *Mathematical Programming Computation*, 4(4):333–361, July 2012. doi: 10.1007/s12532-012-0044-1. URL https://doi.org/10.1007/s12532-012-0044-1.

Zhou, D., Cao, Y., and Gu, Q. Accelerated factored gradient descent for low-rank matrix factorization. In *International Conference on Artificial Intelligence and Statistics*, pp. 4430–4440. PMLR, 2020.

COMP 514 - Project Report
Federated Learning at the Wireless Edge

Nishant Mehrotra [1]

Abstract

Federated learning (FL) is an emerging distributed ML paradigm that trains a model across a large number of mobile devices with the assistance of a central server. The key challenges in FL include large communication overhead and synchronization across devices. However, both these challenges can be efficiently solved by deploying FL over wireless networks a.k.a. wireless FL. In this report, we consider a digital wireless FL setup and design 'channel-aware' local and global model quantization schemes that optimally utilize the available wireless channel resources. We first derive the effect of joint uplink-downlink quantization on the convergence rate of FedAvg, and show that uplink is the primary performance bottleneck in wireless FL. We then design a 'channel-aware' uplink-downlink budget allocation scheme that results in a fair budget allocation across all users.

1. Introduction

Federated learning (FL) (McMahan et al., 2017; Kairouz et al., 2019) is an emerging distributed machine learning (ML) paradigm that trains a ML model across a large number (e.g. $O(10^{10})$) of mobile devices with the assistance of a central server. To ensure user privacy, the model is periodically broadcast by the server to the devices; the devices perform local updates to the model and send model updates back to the server for model aggregation. This is contrast to centralized ML, where users' local datasets are aggregated at the server prior to training/inference.

The typical FL work-flow consists of - (i) local training at the devices, (ii) uplink communication of local model updates from the devices to the central server, (iii) model aggregation and global model updates at the central server, and (iv) downlink communication of global model updates from the central server to the devices. The distributed and iterative nature of the FL work-flow leads to (Kairouz et al., 2019) (a) non-IID local dataset distributions at the devices, (b) model aggregation and synchronization issues at the server, (c) uplink and downlink communication overhead, and (d) large-scale deployment issues.

A possible solution for (b), (c) and (d) above is to deploy FL over wireless networks a.k.a. *wireless FL*. The wireless setting allows for natural 'over-the-air' aggregation due to the inherent broadcast nature of analog wireless transmissions, robustness to synchronization issues (with digital schemes), as well as the opportunity to support a very large number of devices via backhaul networks (e.g. cellular). However, the wireless medium is susceptible to random, time-varying packet drops due to fading and noise. In addition, wireless links have bandwidth and latency constraints that must be satisfied to guarantee reliability (error-free decoding).

Therefore, the key challenge in wireless FL is to design communication-efficient local and global model update schemes that optimally utilize the wireless channel resources while guaranteeing model convergence.

1.1. Prior Work

Prior work on wireless FL can be categorized by the wireless transmission schemes used - (i) digital schemes have high reliability, are inherently robust to synchronization issues, are compatible with commonly used wireless hardware, but require sequential decoding of users' transmissions, (ii) analog schemes allow for natural 'over-the-air' aggregation of simultaneous transmissions, but have low reliability and tight synchronization requirements, and are generally incompatible with commonly used wireless hardware. Although analog schemes generally perform better in terms of model convergence over digital schemes (Mohammadi Amiri & Gündüz, 2020; Amiri & Gündüz, 2020; Sery & Cohen, 2020), in this work we focus on digital schemes due to their compatibility with commonly used wireless hardware.

The bulk of prior work has focused on designing digital uplink quantization and rate allocation schemes for wireless FL; in (Mohammadi Amiri & Gündüz, 2020; Amiri & Gündüz, 2020) a fixed digital uplink quantization scheme

[1]Department of Electrical and Computer Engineering, Rice University, Houston, Texas, USA. Correspondence to: Nishant Mehrotra <nm30@rice.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, Vienna, Austria, 2021.

based on sparsification is presented, and user rate allocation is performed in a multi-user setup by adaptively tuning the users' uplink quantization budgets in a 'channel-aware' manner (Amiri et al., 2020a; Dinh et al., 2020; Chang & Tandon, 2020). Similarly, there has been prior work on designing downlink quantization and model compression schemes, such as in (Khaled & Richtárik, 2020; Yuan et al., 2020), where fixed digital downlink model compression schemes based on sparsification are presented. However, there has been very limited prior work on the more practically relevant *joint* uplink-downlink quantization scenario except for (Amiri et al., 2020b), where a 'channel-aware' downlink rate allocation scheme is presented (with fixed uplink budgets). To the best of our knowledge, there has been no prior work on 'channel-aware' joint uplink-downlink quantization and budget allocation for wireless FL.

1.2. Our Contributions

The main goal of this work is to design 'channel-aware' joint uplink-downlink quantization schemes that allow both the users and the server to allocate optimal quantization budgets depending on the channel conditions to optimally utilize the shared wireless channel. As a first step towards this goal, we first quantify the effect of joint uplink-downlink quantization on the convergence rate of FedAvg (McMahan et al., 2017) for a single-user setup with and without wireless constraints. We show that *uplink* is the primary performance bottleneck, and thus uplink resource allocation must be performed more carefully compared to downlink resource allocation. We verify this insight via simple simulations for the single-user setup. Finally, we consider the more practically relevant multi-user setup with joint uplink-downlink quantization, and present a basic formulation for the 'channel-aware' quantization budget allocation problem.

1.3. Organization of the Report

We first present certain preliminaries and assumptions that we use throughout the report in Section 2. In Section 3, we consider the single-user setup with and without wireless constraints, and present our main theoretical result on the convergence rate of FedAvg with joint uplink-downlink quantization. In Section 4, we extend the model to the multi-user setting and formulate the 'channel-aware' quantization budget allocation problem. We present numerical evaluation of our results in Section 5. Finally, we conclude the report with some directions for future work in Section 6.

1.4. Notation

We use bold uppercase for matrices (e.g. $\mathbf{X}$), bold lowercase for vectors (e.g. $\mathbf{x}$), and non-bold lowercase for scalars (e.g. x). Standard vector and matrix norms are denoted by $\|\cdot\|$ (with appropriate subscripts), and the inner product is denoted by $\langle\cdot,\cdot\rangle$. Other operations, such as the absolute value $|\cdot|$ and the sign function $\operatorname{sgn}(\cdot)$, are assumed to be element-wise unless otherwise mentioned.

2. Preliminaries & Assumptions

2.1. Federated Learning

The goal of federated learning is to collaboratively solve the optimization problem (Khaled & Richtárik, 2020),

$$\mathbf{x}^* \in \arg\min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x}) \right\}, \tag{1}$$

where f_i is the loss function at the ith user, d is the dimension of the model, and n is the number of users. The FedAvg algorithm (McMahan et al., 2017) solves (1) by performing parallel SGD steps at the users followed by a model aggregation step at the central server,

$$\tilde{\mathbf{x}}_{k,i} = \mathcal{Q}_d(\mathbf{x}_k) - \gamma \cdot \nabla f_i(\mathcal{Q}_d(\mathbf{x}_k)), \tag{2}$$

$$\mathbf{x}_{k+1} = \frac{1}{n}\sum_{i=1}^{n} \mathcal{Q}_u^{(i)}(\tilde{\mathbf{x}}_{k,i}), \tag{3}$$

where $\mathcal{Q}_u^{(i)}$ denotes the uplink quantization operator at the ith user, and $\mathcal{Q}_d$ denotes the downlink quantization operator at the central server.

2.2. Quantization

Throughout, we consider the quantization operators to be unbiased and to have bounded variance.

Property 1. *A quantization operator $\mathcal{Q}:\mathbb{R}^d\to\mathbb{R}^d$ is unbiased if*

$$\mathbb{E}_{\mathcal{Q}}[\mathcal{Q}(\mathbf{x})|\mathbf{x}] = \mathbf{x}.$$

Property 2. *A quantization operator $\mathcal{Q}:\mathbb{R}^d\to\mathbb{R}^d$ has bounded variance (with variance factor ω) if*

$$\mathbb{E}_{\mathcal{Q}}\left[\|\mathcal{Q}(\mathbf{x})-\mathbf{x}\|_2^2|\mathbf{x}\right] \le \omega\cdot\|\mathbf{x}\|_2^2.$$

Intuitively, $\omega = 0$ above implies infinite quantization, whereas a large value of ω implies coarse quantization.

Henceforth, we use the following stochastic quantization scheme from (Alistarh et al., 2017; Chang & Tandon, 2020),

$$\mathcal{Q}(x_i) = \|\mathbf{x}\|_2 \cdot \operatorname{sgn}(x_i)\cdot\xi\left(\frac{|x_i|}{\|\mathbf{x}\|_2}, k\right), \tag{4}$$

where k is the number of quantization levels. Given $\frac{x_i}{\|\mathbf{x}\|_2} \in [\frac{l}{k}, \frac{l+1}{k}]$, $\xi\left(\frac{|x_i|}{\|\mathbf{x}\|_2}, k\right)$ is defined as follows,

$$\xi\left(\frac{|x_i|}{\|\mathbf{x}\|_2}, k\right) = \begin{cases} \frac{l}{k} & \text{w.p. } \left(1 + l - \frac{|x_i|}{\|\mathbf{x}\|_2}\cdot k\right) \\ \frac{l+1}{k} & \text{w.p. } \left(\frac{|x_i|}{\|\mathbf{x}\|_2}\cdot k - l\right) \end{cases}. \tag{5}$$

Note that $\mathcal{Q}$ in (4) satisfies Properties 1 and 2 with $\omega = \frac{d}{4k^2}$ (Alistarh et al., 2017; Chang & Tandon, 2020). Also, similar to (Alistarh et al., 2017), we assume that to send $\mathcal{Q}(\mathbf{x})$, the tuple $(\|\mathbf{x}\|_2, \text{sgn}(\mathbf{x}), \xi)$ is sent with $\|\mathbf{x}\|_2$ and $\text{sgn}(\mathbf{x})$ sent using fixed, pre-determined 32 and 2 bit representations respectively. In contrast, variable length encoding using $d \cdot \log_2(k)$ bits is assumed for sending ξ.

2.3. Single-User Wireless Channel Model

We consider an idealized digital channel model that imposes a quantization budget K below which error-free recovery is guaranteed (e.g. using capacity-achieving channel codes), but above which only a noisy recovery is possible,

$$\mathcal{Q}_{ch}(y_i) = \begin{cases} y_i & k \leq K \\ \xi\left([y_i + w_i]_{[0,1]}, k\right) & k > K \end{cases}, \quad (6)$$

where $y_i \in [0,1]$, $w_i \sim \mathcal{N}\left(0, \frac{1}{4K^2}\right)$ is a channel noise term dependent on the quantization budget K, and $[\cdot]_{[0,1]}$ is the clipping operator over the $[0,1]$ interval. Note that we only consider the effect of wireless channels on sending ξ; for simplicity, it is assumed that $\|\mathbf{x}\|_2$ and $\text{sgn}(\mathbf{x})$ are always recovered exactly at the destination. Also, note that on passing $\xi\left(\frac{|x_i|}{\|\mathbf{x}\|_2}, k\right)$ through $\mathcal{Q}_{ch}$, the output becomes unbiased when $k > K$ due to the additive channel noise.

2.4. Multi-User Wireless Channel Model

To model the effect of multiple users transmitting over a shared wireless channel, we consider the well-known multiple access (MAC) and broadcast channel models for uplink and downlink respectively. Given n users, we represent multi-user wireless channels by sum rate constraints over all possible subsets of $\{1, \cdots, n\}$ (Cover & Thomas, 2006),

$$\sum_{i \in \mathcal{S}} d \cdot \log_2\left(k^{(i)}\right) \leq R^{(\mathcal{S})}, \ \mathcal{S} \subseteq \{1, \cdots, n\}, \quad (7)$$

which is equivalent to the following budget constraints,

$$\prod_{i \in \mathcal{S}} k^{(i)} \leq K^{(\mathcal{S})}, \ \mathcal{S} \subseteq \{1, \cdots, n\}. \quad (8)$$

Note that in general, the sum rate constraint for any subset of $\{1, \cdots, n\}$ is smaller than the sum of the constraints for the individual users in the given subset. Thus, the feasible rate region is a convex polytope in general. For example, the feasible region for a two-user setup is shown in Figure 1.

3. Single-User Uplink-Downlink Quantization

We begin by analyzing the convergence guarantees of FedAvg for a single user setup with joint uplink-downlink quantization under two settings - (a) the error-free setting with no quantization budget and no wireless channel noise, and (b) the wireless setting with the channel model discussed in Section 2.3. Note that in the single-user case, the FedAvg iterations in (2) and (3) collapse to gradient descent iterations with uplink and downlink model quantization.

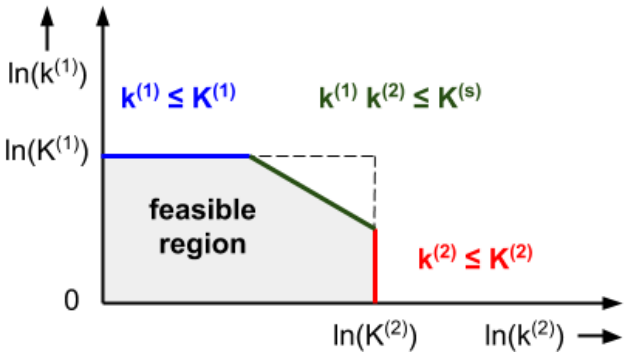

Figure 1. Feasible rate region for two-user wireless channel model.

3.1. Error-Free Setting

In the error-free setting, we obtain the following result -

Theorem 1. *For L-smooth, μ-strongly convex loss functions $f(\cdot)$, single-user FedAvg with constant step-size γ satisfies*

$$\mathbf{r}_k \leq \left(1 - \mu\gamma + 4\omega_u\left(1 + \gamma^2 L^2\right)(1 + 2\omega_d)\right)^k \cdot \mathbf{r}_0 + \frac{2\|\mathbf{x}^*\|_2^2 \cdot (C_1 + C_2)}{\mu\gamma - 4\omega_u(1 + \gamma^2 L^2)(1 + 2\omega_d)},$$

for all valid quantization operators that satisfy

$$\frac{4\omega_u}{\mu} \leq \frac{\gamma}{(1 + \gamma^2 L^2) \cdot (1 + 2\omega_d)},$$
$$\frac{4\omega_d}{\mu} \leq \frac{1 - 2\gamma L}{2\gamma L^2 + \frac{2}{\gamma} + L - \mu},$$

where $\mathbf{r}_{k+1} = \mathbb{E}_{\mathcal{Q}_u, \mathcal{Q}_d}\left[\|\mathbf{x}_{k+1} - \mathbf{x}^*\|_2^2 \,|\mathbf{x}_k, \tilde{\mathbf{x}}_k\right]$ for $\tilde{\mathbf{x}}_k$ defined as in (2), $C_1 = \gamma\omega_d(L - \mu) + 2\omega_d\left(1 + \gamma^2 L^2\right)$ and $C_2 = 2\omega_u + 4\omega_u\omega_d\left(1 + \gamma^2 L^2\right)$.

Proof. See Appendix A. □

Corollary 1. *In contrast to downlink-only model compression/quantization schemes (Khaled & Richtárik, 2020; Yuan et al., 2020) that satisfy the following guarantees,*

$$\mathbf{r}_k \leq (1 - \mu\gamma)^k \cdot \mathbf{r}_0 + \frac{2\|\mathbf{x}^*\|_2^2 \cdot C_1}{\mu\gamma},$$

FedAvg with joint uplink-downlink quantization converges at a slower rate (by a factor of $4\omega_u\left(1 + \gamma^2 L^2\right)(1 + 2\omega_d)$)

to a larger error neighborhood (due to the additive term C_2 in the numerator and the subtractive term $4\omega_u\left(1+\gamma^2L^2\right)(1+2\omega_d)$ in the denominator).

Note that on substituting $\omega_u = 0$ in Theorem 1, we readily obtain Corollary 1; thus, our results are more general compared to (Khaled & Richtárik, 2020; Yuan et al., 2020).

Theorem 1 and Corollary 1 imply that *uplink* is the primary performance bottleneck in wireless FL system designs. For a numerical comparison between uplink-only versus downlink-only quantization, consider the parameter values $\gamma = \frac{1}{4L}$ and $\omega_u = \omega_d = \frac{1}{73\kappa}$, where $\kappa = \frac{L}{\mu}$ is the condition number; while a convergence rate of $O\left(4\kappa \cdot \ln\left(\frac{1}{\epsilon}\right)\right)$ to an error neighborhood of size $O\left(\frac{4\|\mathbf{x}^*\|_2^2}{1387}\right)$ is achieved with downlink-only quantization, with uplink-only quantization we achieve slower convergence at a rate $O\left(5.2\kappa \cdot \ln\left(\frac{1}{\epsilon}\right)\right)$ to a much larger error neighborhood of size $O\left(\frac{2\|\mathbf{x}^*\|_2^2}{7}\right)$.

However, we note that the conditions in Theorem 1 imply more stringent downlink quantization constraints as compared to uplink i.e. in general, finer downlink quantization is required compared to uplink to satisfy the conditions in Theorem 1. We further explore this implication in the multi-user 'channel-aware' budget allocation problem formulation.

3.2. Wireless Setting

For the stochastic quantization scheme in Section 2.2 and all valid uplink and downlink quantization budgets K_U and K_D that satisfy the conditions in Theorem 1, in the wireless setting we make the following preliminary observations -

- When $k_u \leq K_U$ and $k_d \leq K_D$, we expect Theorem 1 with $\omega_u = \frac{d}{4k_u^2}$ and $\omega_d = \frac{d}{4k_d^2}$ to always hold. Here, the available channel resources are under-utilized.
- When $k_u > K_U$ and $k_d > K_D$, we expect Theorem 1 with $\omega_u = \frac{d}{4K_U^2}$ and $\omega_d = \frac{d}{4K_D^2}$ to serve as a lower bound on the sample complexity. Here, the available channel resources are over-utilized.

4. Multi-User 'Channel-Aware' Quantization

In an actual wireless FL deployment with multiple users, it is important to allocate uplink and downlink quantization budgets to users and the server prior to every FedAvg iteration to alleviate the communication overhead while also optimally utilizing the available wireless channel resources. In the context of the multi-user wireless channel model presented in Section 2.4, this corresponds to solving for the individual users' quantization budgets $k^{(i)},\ i \in \{1, \cdots, n\}$ in (8) for both uplink and downlink. In the wireless FL setting, we formulate the 'channel-aware' quantization budget allocation problem as

$$\min_{\{k_u^{(i)},k_d^{(i)}\}_{i=1}^n} \sum_{i=1}^{n} \left[\frac{2\left\|\mathbf{x}^*\right\|_2^2 \cdot \left(C_1^{(i)} + C_2^{(i)}\right)}{\mu\gamma - 4\omega_u^{(i)}\left(1+\gamma^2L^2\right)\left(1+2\omega_d^{(i)}\right)}\right] \tag{9}$$

$$\text{s.t.} \prod_{i\in\mathcal{S}} k_{u/d}^{(i)} \leq K_{U/D}^{(\mathcal{S})},\ \mathcal{S} \subseteq \{1, \cdots, n\}, \tag{10}$$

where from among all feasible budgets, the optimal uplink and downlink budgets are found such that the sum of the error neighborhoods over all users is minimized. Note that in addition to the sum rate constraints in (10), the quantization operator conditions in Theorem 1 must also be satisfied; as per the discussion in Section 3.1, the feasible user budget region is expected to be larger for uplink versus downlink.

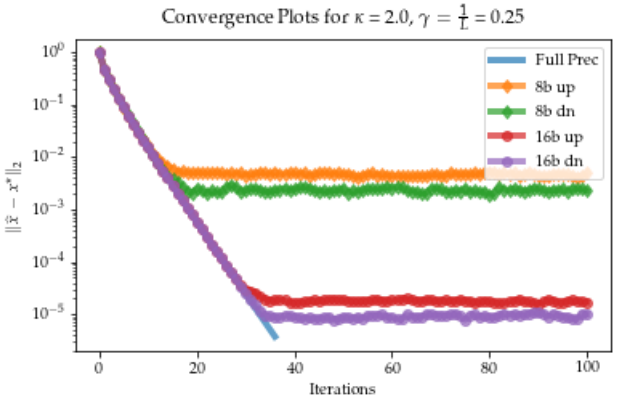

Figure 2. Convergence plots for single-user error-free FedAvg.

5. Numerical Evaluation

5.1. Single-User Uplink-Downlink Quantization

We begin by comparing the performance of single-user uplink-only versus downlink-only quantization in the error-free and wireless settings. As per Theorem 1, we expect linear convergence to a larger error neighborhood around the optimal solution with uplink-only versus downlink-only quantization. Throughout, we simulate a well-conditioned linear regression problem with $\kappa = 2$.

5.1.1. Error-Free Setting

In the error-free setting, we plot the convergence plots for full-precision, 8 bit and 16 bit uplink-only and downlink-only quantization in Figure 2. As expected from Theorem 1, we observe a linear convergence to an error neighborhood dependent on the number of quantization levels. For example, with downlink-only quantization we observe an error neighborhood difference of $O\left(\log_{10}\left(2^8\right)\right) =$

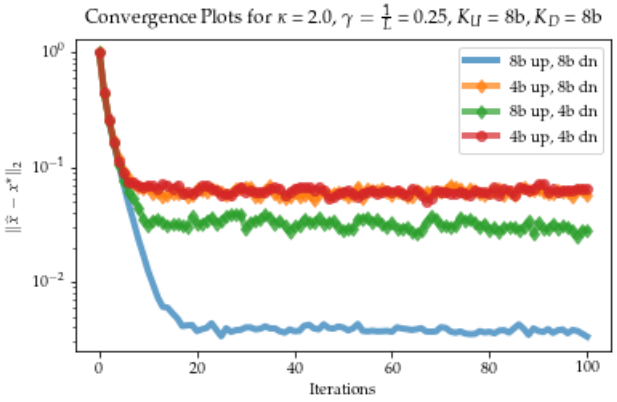

Figure 3. Convergence plots for single-user wireless FedAvg when $k_{u/d} \leq K_{U/D}$.

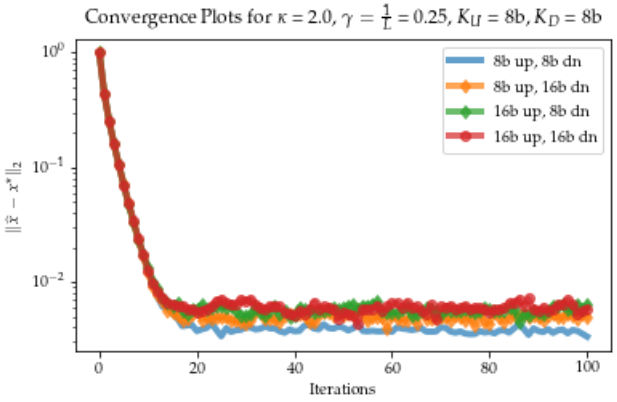

Figure 4. Convergence plots for single-user wireless FedAvg when $k_{u/d} > K_{U/D}$.

$O\left(\log_{10}\left(\sqrt{\kappa\omega_d}|_{k_d=2^8}\right)\right) - O\left(\log_{10}\left(\sqrt{\kappa\omega_d}|_{k_d=2^{16}}\right)\right)$ between the 8 bit and 16 bit downlink-only plots. We also observe an increase in the error neighborhood size with uplink-only versus downlink-only quantization as expected.

5.1.2. Wireless Setting

In the wireless setting, we plot the convergence plots for uplink and downlink quantization budgets of $K_U = 8$ bits and $K_D = 8$ bits in Figures 3 and 4. We compare the performance of uplink-only and downlink-only quantization when (i) the user's quantization budget is smaller than the link's available budget i.e. $k_{u/d} \leq K_{U/D}$, and (ii) the user's quantization budget exceeds the link's available budget i.e. $k_{u/d} > K_{U/D}$.

When $k_{u/d} \leq K_{U/D}$, as per Section 3.2 we expect linear convergence to an error neighborhood dependent on $k_{u/d}$. In Figure 3, we plot the convergence plots for all four combinations of 4 bit and 8 bit uplink and downlink quantization. We observe that the best performance is achieved when $k_{u/d} = K_{U/D}$ i.e. the available channel resources are optimally utilized. In addition, we observe that the performance deteriorates a lot more on under-utilizing the uplink channel versus under-utilizing the downlink channel. This reinforces our conclusion from Theorem 1 that uplink is the primary performance bottleneck in wireless FL; therefore, uplink channel resources need to always be optimally utilized.

When $k_{u/d} > K_{U/D}$, as per Section 3.2 we expect worsening performance as the link budgets $K_{U/D}$ are exceeded. In Figure 4, we plot the convergence plots for all four combinations of 8 bit and 16 bit uplink and downlink quantization. We observe that the best performance is achieved when $k_{u/d} = K_{U/D}$ i.e. the available channel resources are optimally utilized, and that the performance deteriorates by an amount proportional to the additive channel noise in Section 2.3 on over-utilizing the channels. In addition, we observe that performance deteriorates more on over-utilizing the uplink channel versus over-utilizing the downlink channel. This reinforces our conclusion from Theorem 1 that uplink is the primary performance bottleneck in wireless FL. On comparing Figures 3 and 4, we also observe that under-utilization of channel resources leads to a greater performance loss compared to over-utilization of channel resources. Thus, optimal allocation of user budgets is of great relevance to practical wireless FL system deployment.

5.2. Multi-User 'Channel-Aware' Quantization

Finally, we solve the 'channel-aware' budget allocation problem formulated in Section 4 for two users with uplink-only and downlink-only quantization. For brevity, we solve only the uplink-only quantization problem analytically, for which the Lagrangian function takes the form

$$\mathcal{L} = J\left(k_u^{(1)}, k_u^{(2)}\right) + \lambda_s \cdot \left(\prod_{i\in\{1,2\}} k_u^{(i)} - K_U^{(s)}\right) + \lambda_1 \cdot \left(k_u^{(1)} - K_U^{(1)}\right) + \lambda_2 \cdot \left(k_u^{(2)} - K_U^{(2)}\right), \quad (11)$$

where $J\left(k_u^{(1)}, k_u^{(2)}\right)$ is the objective function in (9) i.e. $J\left(k_u^{(1)}, k_u^{(2)}\right) = \sum_{i=1}^{2}\left[\frac{2\|\mathbf{x}^*\|_2^2 \cdot \left(C_1^{(i)} + C_2^{(i)}\right)}{\mu\gamma - 4\omega_u^{(i)}(1+\gamma^2L^2)\left(1+2\omega_d^{(i)}\right)}\right]$. From the rate region in Figure 1, it is clear that the constraint $\prod_{i\in\{1,2\}} k_u^{(i)} \leq K_U^{(s)}$ must be satisfied with equality in order to result in maximal channel utilization (Cover & Thomas, 2006; Chang & Tandon, 2020). However, this implies that the individual constraints $k_u^{(i)} \leq K_U^{(i)}$, $i \in \{1,2\}$ aren't satisfied with equality; thus, $\lambda_1 = \lambda_2 = 0$. Thus, the optimal budgets are found by setting the partial derivatives of the Lagrangian with respect to the user budgets

Table 1. Optimal uplink and downlink two-user budget allocation.

$\left(K_{U/D}^{(1)}, K_{U/D}^{(2)}, K_{U/D}^{(s)}\right)$	UP $\left(k_u \geq 2^{5.37}\right)$	DN $\left(k_d \geq 2^{6.43}\right)$
$(2^9, 2^6, 2^{14})$	$(2^8, 2^6)$	-
$(2^9, 2^7, 2^{14})$	$(2^7, 2^7)$	$(2^7, 2^7)$
$(2^9, 2^8, 2^{14})$	$(2^7, 2^7)$	$(2^7, 2^7)$
$(2^9, 2^9, 2^{14})$	$(2^7, 2^7)$	$(2^7, 2^7)$
$(2^9, 2^{10}, 2^{14})$	$(2^7, 2^7)$	$(2^7, 2^7)$

$k_u^{(i)},\ i \in \{1, 2\}$ to zero and solving for the variable λ_s. This results in

$$\frac{k_u^{(1)}}{k_u^{(2)}} = \left(\frac{\mu\gamma - 4\omega_u^{(2)}\left(1+\gamma^2 L^2\right)\left(1+2\omega_d^{(2)}\right)}{\mu\gamma - 4\omega_u^{(1)}\left(1+\gamma^2 L^2\right)\left(1+2\omega_d^{(1)}\right)}\right) \times \sqrt{\frac{\mu\gamma + 4\omega_u^{(1)}\left(1+\gamma^2 L^2\right)\left(1+2\omega_d^{(1)}\right)}{\mu\gamma + 4\omega_u^{(2)}\left(1+\gamma^2 L^2\right)\left(1+2\omega_d^{(2)}\right)}}, \quad (12)$$

which in conjuction with $\prod_{i\in\{1,2\}} k_u^{(i)} = K_U^{(s)}$ results in the optimal user budget allocation $\{k_u^{(1)}, k_u^{(2)}\}$. Note that in addition to the constraints in (10), the quantization operator conditions in Theorem 1 must also be satisfied; as per the discussion in Section 4, we expect a larger feasible user budget region for uplink as compared to downlink.

We numerically evaluate the optimal uplink-only and downlink-only user budgets for the two-user setup; the results are tabulated in Table 1. For simplicity, we keep the user 1 and joint link budgets constant at $K_{U/D}^{(1)} = 2^9$ and $K_{U/D}^{(s)} = 2^{14}$ respectively and vary user 2's link budget from 2^6 to 2^{10} in order to quantify the effect of varying link budgets on the optimal user budget values. We observe that the optimal user budgets are allocated fairly across both users to the value $\sqrt{K_{U/D}^{(s)}} = 2^7$ for all link budget values that satisfy $K_{U/D}^{(i)} \geq \sqrt{K_{U/D}^{(s)}} = 2^7,\ i \in \{1, 2\}$. Otherwise, the feasible pair closest to the maximally fair pair $\left(\sqrt{K_{U/D}^{(s)}}, \sqrt{K_{U/D}^{(s)}}\right) = (2^7, 2^7)$ gets allocated. In addition, as expected, we note that the feasible user budget region is larger for uplink as compared to downlink.

6. Concluding Remarks

In this report, we considered the design of 'channel-aware' quantization schemes for wireless FL. We first extend prior convergence analysis of FedAvg from the downlink-only model compression/quantization setting to joint uplink-downlink quantization; for a single-user setup, our results demonstrate that uplink is the primary performance bottleneck in wireless FL system designs. Therefore, uplink channels must be maximally utilized in a practical wireless FL system design to result in the best system performance. However, an advantage of joint uplink-downlink quantization is that the uplink quantization constraints are less stringent compared to the corresponding downlink quantization constraints outlined in prior work. Therefore, in a multi-user wireless FL system, the feasible user budget region is larger for uplink as compared to downlink. We verify this insight by solving for the optimal 'channel-aware' user quantization budgets in uplink and downlink; in addition to a larger uplink feasible user budget region, we demonstrate that for the stochastic quantization scheme we use in this work, 'channel-aware' quantization results in a fair user budget allocation across all users in uplink and downlink.

Some drawbacks of our current setup are that (i) FedAvg converges only to a large error neighborhood around the optimal solution with joint uplink-downlink quantization, and (ii) our 'channel-aware' user budget allocation problem formulation does not take into account the 'informativeness' of the users' local model updates. As part of future work, we intend to (i) derive generic lower convergence bounds across a large class of wireless FL algorithms (similar to Nesterov's ower bounds for convex optimization), (ii) jointly design the optimization algorithms (e.g. FedAvg) along with the wireless access schemes (e.g. uplink-downlink quantization operators) that achieve the lower bounds derived in (i), and (iii) design more generic quantization schemes (similar to (Chang & Tandon, 2020)) that take the 'informativeness' of users' local model updates into account. In addition, from a more theoretical perspective, it would be interesting to explore whether insights from lossy compression in information theory (rate-distortion theory) can be applied to the design of wireless FL update schemes.

Acknowledgements

We thank Chen Dun and Cameron Wolfe for discussions.

References

Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30, pp. 1709–1720. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/6c340f25839e6acdc73414517203f5f0-Paper.pdf.

Amiri, M. M. and Gündüz, D. Federated Learning Over Wireless Fading Channels. *IEEE Transactions on Wireless Communications*, 19(5):3546–3557, 2020.

Amiri, M. M., Gunduz, D., Kulkarni, S. R., and Poor, H. V. Convergence of Update Aware Device Scheduling for Federated Learning at the Wireless Edge. May 2020a. URL http://arxiv.org/abs/2001.10402.

Amiri, M. M., Gunduz, D., Kulkarni, S. R., and Poor, H. V. Convergence of Federated Learning over a Noisy Downlink. Aug 2020b. URL http://arxiv.org/abs/2008.11141.

Chang, W.-T. and Tandon, R. Communication Efficient Federated Learning over Multiple Access Channels. Aug 2020. URL http://arxiv.org/abs/2001.08737.

Cover, T. M. and Thomas, J. A. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, USA, 2006. ISBN 0471241954.

Dinh, C., Tran, N. H., Nguyen, M. N. H., Hong, C. S., Bao, W., Zomaya, A. Y., and Gramoli, V. Federated Learning over Wireless Networks: Convergence Analysis and Resource Allocation. March 2020. URL http://arxiv.org/abs/1910.13067.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R. G. L., Rouayheb, S. E., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P. B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konečný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Raykova, M., Qi, H., Ramage, D., Raskar, R., Song, D., Song, W., Stich, S. U., Sun, Z., Suresh, A. T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F. X., Yu, H., and Zhao, S. Advances and Open Problems in Federated Learning. Dec 2019. URL http://arxiv.org/abs/1912.04977.

Khaled, A. and Richtárik, P. Gradient Descent with Compressed Iterates. March 2020. URL http://arxiv.org/abs/1909.04716.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-Efficient Learning of Deep Networks from Decentralized Data. volume 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR. URL http://proceedings.mlr.press/v54/mcmahan17a.html.

Mohammadi Amiri, M. and Gündüz, D. Machine Learning at the Wireless Edge: Distributed Stochastic Gradient Descent Over-the-Air. *IEEE Transactions on Signal Processing*, 68:2155–2169, 2020.

Sery, T. and Cohen, K. On Analog Gradient Descent Learning Over Multiple Access Fading Channels. *IEEE Transactions on Signal Processing*, 68:2897–2911, 2020.

Yuan, B., Dun, C., Kyrillidis, A., and Jermaine, C. M. Distributed Learning of Neural Networks using Independent Subnet Training. June 2020. URL http://arxiv.org/abs/1910.02120.

A. Proof of Theorem 1

We follow the general proof strategy in (Khaled & Richtárik, 2020); for brevity, we only outline the key arguments that differ in our setting as compared to theirs.

Proof. Let $\mathbf{r}_{k+1} = \mathbb{E}_{\mathcal{Q}_u,\mathcal{Q}_d}\left[\|\mathbf{x}_{k+1}-\mathbf{x}^*\|_2^2\,|\mathbf{x}_k,\tilde{\mathbf{x}}_k\right]$, where $\tilde{\mathbf{x}}_k = \mathcal{Q}_d\left(\mathbf{x}_k\right) - \gamma\cdot\nabla f\left(\mathcal{Q}_d\left(\mathbf{x}_k\right)\right)$. We have,

$$\|\mathbf{x}_{k+1}-\mathbf{x}^*\|_2^2 = \|\mathcal{Q}_u\left(\tilde{\mathbf{x}}_k\right)-\mathbf{x}^*\|_2^2 \tag{13}$$

$$= \|\mathcal{Q}_u\left(\tilde{\mathbf{x}}_k\right)-\tilde{\mathbf{x}}_k+\tilde{\mathbf{x}}_k-\mathbf{x}^*\|_2^2, \tag{14}$$

which on expanding, taking expectations with respect to $\mathcal{Q}_u$ and $\mathcal{Q}_d$, and using Property 1 results in

$$\mathbf{r}_{k+1} = \mathbb{E}_{\mathcal{Q}_u,\mathcal{Q}_d}\left[\|\mathcal{Q}_u\left(\tilde{\mathbf{x}}_k\right)-\tilde{\mathbf{x}}_k\|_2^2\,|\mathbf{x}_k,\tilde{\mathbf{x}}_k\right] + \mathbb{E}_{\mathcal{Q}_d}\left[\|\tilde{\mathbf{x}}_k-\mathbf{x}^*\|_2^2\,|\mathbf{x}_k\right]. \tag{15}$$

The second term in (15) may be upper bounded via the main results of (Khaled & Richtárik, 2020; Yuan et al., 2020),

$$\mathbb{E}_{\mathcal{Q}_d}\left[\|\tilde{\mathbf{x}}_k-\mathbf{x}^*\|_2^2\,|\mathbf{x}_k\right] \leq (1-\mu\gamma)\cdot\mathbf{r}_k + 2\,\|\mathbf{x}^*\|_2^2\cdot C_1, \tag{16}$$

and to upper bound the first term in (15), we use Property 2,

$$\mathbb{E}_{\mathcal{Q}_u,\mathcal{Q}_d}\left[\|\mathcal{Q}_u\left(\tilde{\mathbf{x}}_k\right)-\tilde{\mathbf{x}}_k\|_2^2\,|\mathbf{x}_k,\tilde{\mathbf{x}}_k\right] \leq \omega_u\cdot\mathbb{E}_{\mathcal{Q}_d}\left[\|\tilde{\mathbf{x}}_k\|_2^2\,|\mathbf{x}_k\right]. \tag{17}$$

Via the triangle inequality, Lemma 3 in (Khaled & Richtárik, 2020) (with $\mathbf{y}=\mathbf{x}^*$), and Property 2, we have

$$\|\tilde{\mathbf{x}}_k\|_2^2 \leq 2\cdot\left(\|\tilde{\mathbf{x}}_k-\mathbf{x}_k\|_2^2 + 2\cdot\|\mathbf{x}_k-\mathbf{x}^*\|_2^2 + 2\cdot\|\mathbf{x}^*\|_2^2\right), \tag{18}$$

$$\mathbb{E}_{\mathcal{Q}_d}\left[\|\tilde{\mathbf{x}}_k-\mathbf{x}_k\|_2^2\,|\mathbf{x}_k\right] \leq 4\omega_d\left(1+\gamma^2L^2\right)\cdot\|\mathbf{x}^*\|_2^2 + \left(2\gamma^2L^2+4\omega_d\left(1+\gamma^2L^2\right)\right)\cdot\|\mathbf{x}_k-\mathbf{x}^*\|_2^2. \tag{19}$$

From (15), (16), (17), (18) and (19), we obtain

$$\mathbf{r}_{k+1} \leq \left(1 - \mu\gamma + 4\omega_u \left(1 + \gamma^2 L^2\right)(1 + 2\omega_d)\right) \cdot \mathbf{r}_k + 2\left\|\mathbf{x}^*\right\|_2^2 \cdot (C_1 + C_2), \quad (20)$$

where $C_1 = \gamma\omega_d (L - \mu) + 2\omega_d \left(1 + \gamma^2 L^2\right)$ and $C_2 = 2\omega_u + 4\omega_u\omega_d \left(1 + \gamma^2 L^2\right)$. On unrolling (20), we obtain the desired result. Note that for (20) to behave as a contraction, we require $4\omega_u \left(1 + \gamma^2 L^2\right)(1 + 2\omega_d) \leq \mu\gamma$ which directly leads to the uplink quantization operator condition in Theorem 1. The downlink quantization operator condition follows from the discussion in (Khaled & Richtárik, 2020) leading to (16). □

A Review of Low-Rank Solutions to Semidefinite Programming

Abdelrahman Abouzeid, Shawn Fan

Fall 2020

1 Introduction

1.1 Definition of SDP

A semidefinite program (SDP) is an optimization problem of the form

$$\begin{aligned} &\min \langle C, X \rangle \\ \text{s.t. } &\langle A_i, X \rangle = b_i, i = 1, \ldots, m \\ &X \succeq 0 \end{aligned}$$

The optimization variable is $X \in \mathbb{S}^n$,, where $\mathbb{S}^n$ denotes the set of all $n \times n$ symmetric matrices, and the problem data are $A_1, \ldots, A_m, C \in \mathbb{S}^n$ and $b \in \mathbb{R}^m$.

The dual problem of SDP is

$$\begin{aligned} &\max b^\top y \\ \text{s.t. } &\sum_{i=1}^{m} y_i A_i + S = C \\ &S \succeq 0 \end{aligned}$$

The optimization variables are S and y.

1.2 Applications of SDP

1.2.1 Combinatorial optimization: The MAX CUT problem

SDP has wide applicability in combinatorial optimization. A number of NP-hard combinatorial optimization problems have convex relaxations that are semidefinite programs. The MAXCUT problem is an example that demonstrates the use of SDP in combinatorial optimization [10].

Let G be an undirected graph with node set $N = \{1, \ldots, n\}$ and edge set E. Let $w_{ij} = w_{ji}$ e the weight of edge (i, j), for $(i, j) \in E$. Assuming that $w_{ij} > 0$ for all $(i, j) \in E$,, the MAX CUT problem aims to determine a subset S of the nodes N for which the sum of the weights of the edges

that cross from S to its complement $N \setminus S$. Let $x_j = 1$ for $j \in S$ and $x_j = -1$ for $j \in N \setminus S$. The MAX CUT problem can be formulated as

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(1 - x_i x_j)$$
$$\text{s.t. } x \in \{-1, 1\}, j = 1, \ldots, n$$

Let $Y = xx^\top$ and W be a matrix whose $(i,j)^{\text{th}}$ entry is w_{ij}. The MAX CUT problem can be equivalently formulated as

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} - \langle W, Y \rangle$$
$$\text{s.t. } Y = xx^\top, Y_{jj} = 1, j = 1, \ldots, n$$

The matrix $Y = xx^\top$ is a symmetric rank-1 positive semidefinite matrix. By removing the rank-1 restriction, we can relax the problem into the following semidefinite program [10].

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} - \langle W, Y \rangle$$
$$\text{s.t. } Y \succeq 0, Y_{jj} = 1, j = 1, \ldots, n$$

The MAX CUT problem has direct applications in modern machine learning. For example, in graph-based clustering, each node in a fully-connected graph is a data point. The weight of each edge is some measure of dissimilarity between data points. Solving the MAX CUT problem is equivalent to bipartitioning the data and maximizing inter-cluster distance.

1.2.2 Low-Rank Matrix Completion

SDP also has a variety of applications in Low-Rank Matrix Completion. One notable application is the Netflix Problem which is given a matrix $Z \in \mathbb{R}^{mxn}$, with rows i = [1, 2, ..., n] corresponding to users and columns j = [1, 2, ..., m] corresponding to shows, the matrix represents what rating user i gives to user j, the matrix is partially incomplete (some entries needs to be filled) the task then is to infer what are the values of the missing entries. Given the current set-up any value is valid for the missing entries, so another constraint is added which is the assumption that user base satisfy a finite number of segments k (s.t k ¡¡n) Hence we can assume that the Matrix Z has a low-rank and our problem is defined formally as:

$$\min_{X \in \mathbb{R}^{m \times n}} \sum_{i,j} (X_{i,j} - Z_{i,j})^2$$
$$\text{s.t. } rank(X) \leq r$$

Such that Z is the matrix with data we already know. The indices i,j range over this data we already

know. The problem can then be reduced to an SDP problem by relaxing the constraint on the rank, and changing it into a constraint on the Nuclear Norm. This is defined formally as:

$$\min_{X\in\mathbb{R}^{mxn},W_1\in S_m,W_2\in S_n} \sum_{i,j}(X_{i,j}-Z_{i,j})^2$$
$$\text{s.t. } Tr\begin{bmatrix}W_1 & X\\ X^T & W_2\end{bmatrix}\leq r, \begin{bmatrix}W_1 & X\\ X^T & W_2\end{bmatrix}\succeq 0$$

The Low-Rank Matrix Completion problem has many other applications such as IoT Locatlization: The problem arises in IoT sensor networks. The goal is to recover the sensor map in Euclidean space from a set of pairwise distances. Thus it is an instance of a matrix completion problem with rank n if the sensors are located in an n-Dimensional Space (Usually n = 2, or 3 hence it is actually an instance of Low-Rank Matrix completion)

1.3 Low-Rank Solutions to SDP

This review focuses on low-rank solutions to SDP. Some cases such as the Netflix problem constrain feasible solutions to low-rank matrices. However, in some other problems, low-rankness is not an explicit constraint, and the optimal solution to the SDP problem may in fact have high rank. In these cases, methods of finding low-rank solutions to SDP are still relevant, as it has been shown that if an SDP has m constraints and an optimal solution, then it also has a rank-r optimal solution [8] with

$$\frac{r(r+1)}{2}\leq m$$

For an SDP problem without low-rank constraints, searching for a low-rank optimal solution can lead to significant gains in computational time and storage without severely compromising the accuracy of the solution.

2 Burer-Monteiro based methods

A $n\times n$ positive semi-definite matrix X has rank $r\leq n$ if and only if it can be written as $X=VV^\top$, where $V\in\mathbb{R}^{n\times r}$. The key idea of Burer-Monteiro based methods is to factor the decision variable X as $VV^\top$ where $V\in\mathbb{R}^{n\times r}$. Using this factorization, we can rewrite as

$$\min_{V\in\mathbb{R}^{n\times r}} \text{Tr}(CVV^\top)$$
$$\text{s.t. Tr}(A_iVV^\top)=b_i, i=1,\cdots,m$$

2.1 Augmented Lagrangian method

While the factorization $X=VV^\top$ effectively eliminated the constraint $X\succeq 0$, the difficult constraints $\text{Tr}(A_iVV^\top)=b_i$ remain [2]. The augmented Lagrangian method account for them by ignoring the constraints all together and including additional terms that penalize infeasible points [2]. Penalization alone, however, can lead to ill-conditioning in the optimization, so another feature of the augmented Lagrangian method is the introduction of Lagrange multipliers, one for each

constraint [2]. The augmented Lagrangian object function is given by

$$\mathcal{L}(R, y, \sigma) = \langle C, VV^\top \rangle - \sum_{i=1}^{m} y_i(\langle A_i, VV^\top \rangle - b_i) + \frac{\sigma}{2} \sum_{i=1}^{m} (\langle A_i, VV^\top \rangle - b_i)^2$$

For an appropriate, fixed choice (y^*, σ^*), finding an optimal solution R^* is equivalent to minimizing $\mathcal{L}(R, y^*, \sigma^*)$ with respect to R only. To determine a suitable (y^*, σ^*), the augmented Lagrangian algorithm alternates minimizing $\mathcal{L}(R, y, \sigma)$ over R and (y, σ). This is done by minimizing $\mathcal{L}(R, y^k, \sigma^k)$ with respect to R to find its optimal solution R^k and then using (R^k, y^k, σ^k) to determine a new pair (y^k, σ^k) at each iteration.

Algorithm 1: Augmented Lagrangian Algorithm

compute $v = \sum_{i=1}^{m} (\langle A_i, VV^\top \rangle - b_i)^2$
if $v < \mu v$ **then**
  $y_i^{k+1} = y_i^k - \sigma_k(\langle A_i, VV^\top \rangle - b_i)$, for all i;
  $\sigma_{k+1} = \sigma$;
  $v_{k+1} = v_k$
else
  $y_i^{k+1} = y_i^k$ for all i;
  $\sigma_{k+1} = \gamma\sigma$;
  $v_{k+1} = v_k$

A key component of the algorithm is performing the unconstrained minimization of $\mathcal{L}(R, y^k, \sigma^k)$ with respect to R. In the original study, the authors used a quasi-Newton's method, limited-memory BFGS algorithm [2]. There is no theoretical guarantee regarding convergence to global minima. the algorithm is observed to experimentally return the global minimum and reported strong computational results, a speed-up factor of nearly 500 over the second fastest algorithm at the time, based on the fact that the function and gradient evaluations of the augmented Lagrangian function [2].

Later, the authors developed a perturbed augmented Lagrangian algorithm and proved theoretical convergence [3]. The perturbed algorithm is formulated as

$$\min_{R \in \mathbb{R}^{n \times r}} \mathcal{L}(R, y, \sigma) + \mu \det(R^\top R)$$

where $\{\mu_k\} \subset \mathbb{R}^+$ is a sequence converging to 0. The extra term $\mu \det(R^\top R)$ is needed for theoretical convergence but not for practical convergence [3]. One may theoretically choose $\mu_k > 0$ as small as one wishes, with the only exception being that $\mu_k \to 0$ [3].

2.2 Block-coordinate maximization method

Coordinate-descent methods were originally proposed to solve SDP instances with constraints only on the diagonal entries of the matrix:

$$\min_{X \succeq 0} \langle C, X \rangle$$
$$\text{s.t. } X_{ii} = 1, i = 1, \cdots, n$$

Applying low-rank factorization,

$$\begin{aligned} &\min \langle C, VV^\top \rangle \\ &\text{s.t. } \|v\|_2 = 1, i = 1, \cdots, n \end{aligned}$$

Here we focus on unit diagonal constraints, the methods we discuss can be easily extended to arbitrary positive numbers. Coordinate or block-coordinate descent methods (BCD) select a subset of variables to update at each iteration, reducing the computational complexity at each iteration.

2.2.1 The Mixing Method

The mixing method is one of the earliest block-coordinate maximization algorithm for solving SDP [9]. The mixing method applies a coordinate power iteration routine, updating each column of V in sequence [9]. The objective terms that depend on v_i are given by $v_i^\top (\sum_{j=1}^n C_{ij} v_j)$. Since $\|v\|_2 = 1$, we can assume that $C_{ii} = 0$ without affecting the solution of the problem. Thus, the problem is equivalent to simply minimizing the inner product $v_i^\top g_i$, where $g_i = \sum_{j=1}^n C_{ij} v_j$, subject to the constraint that $\|v\|_2 = 1$. This update is given by

$$v_i := \frac{-\sum_{j=1}^n C_{ij} v_j}{\| \sum_{j=1}^n C_{ij} v_j \|_2}$$

Theoretical analysis of the mixing method constitutes four main convergence properties [9]:

- The Mixing Method is strictly decreasing in objective value and always converges to a first-order critical point over iterations.
- A variant of the Mixing Method with a proper step size converges to a global optimum almost surely under random initialization without any assumptions
- The Mixing Method converges linearly to the global optimum when the solution is close enough, regardless of the existence of nearby non-optimal critical points.
- For a rank $k > \sqrt{2n}$, all non-optimal critical points $V \in \mathbb{R}^{k \times n}$ are unstable for the Mixing Method.

2.2.2 Coordinate selection rules

In the original mixing method, the blocks $v_{i_k}^k$ that are updated at each iteration are chosen deterministically, as each block is updated in sequence. There are three coordinate selection rules, two of which are randomized [4].

- Uniform sampling: $i_k = i$ with probability $p_i = 1/n$
- Importance sampling: $i_k = i$ with probability $p_i = \frac{\|g_i^k\|}{\sum_{j=1}^n \|g_j^k\|}$
- Greedy coordinate selection: $i_k = \arg\max_i (\|g_i^k\| - \langle v_i^k, g_i^k \rangle)$

The greedy selection rule yields the following global convergence guarantee [4]

$$\min_{k\in[K-1]} \|\nabla f(V^k)\|_F^2 \leq \frac{2n\|A\|_1(f(V^*) - f(V^0))}{K}$$

.
The randomized selection rules yield a convergence guarantee as the following [4]

$$\min_{k\in[K-1]} \mathbb{E}\|\nabla f(V^k)\|_F^2 \leq \frac{2L(f(V^*) - f(V^0))}{K}$$

where $L = n\|A\|_1$ for uniform sampling and $L = \|A\|_1$ for importance sampling. Hence, uniform sampling attains the same sublinear convergence rate in expectation as greedy coordinate selection, and importance sampling enjoys a higher convergence rate [4].

2.2.3 Adding momentum to the mixing method

The classical technique for momentum acceleration is the heavy ball method:

$$w_{t+1} = w_t - \eta\nabla f(w_t) + \beta(w_t - w_{t-1})$$

where w represents the variable vector, f is a differentiable loss function, η is the learning rate, and β is the momentum term. However, in power iteration, there is no notion of step size, and the update rule is oblivious to the previous estimate location w_{t-1}. To account for these differences, the following update rule has been used to incorporate momentum in a power iteration algorithm for PCA:

$$w_{t+1} = \text{normalize}(Cw_t - \beta w_{t-1})$$

Naively adapting this update rule for the mixing method:

$$v_i = \text{normalize}(\sum_{i=1}^{n} C_{ij}v_j - \beta v_i)$$

However, v_i has a norm of 1 but $\sum_{i=1}^{n} C_{ij}v_j$ may have a norm much greater than 1. The contribution of the momentum term may end up infinitesimal. To ensure that $\sum_{i=1}^{n} C_{ij}v_j$ and v_i are comparable in magnitude, an additional projection step is added, resulting in the Mixing Method++. The Mixing Method++ has similar convergence rate as the original Mixing Method in theory, but in practice, the Mixing Method++ provides significant speed-up.

3 Frank-Wolfe based methods

An $n \times n$ positive semi-definite matrix X has rank $r \leq n$ if and only if it can be written as $X = VV^\top$, where $V \in \mathbb{R}^{n\times r}$. The key idea of Frank-Wolfe is it minimizes a constrained function resulting in an optimization problem of the form $\min_{x\in S} f(x)$ such that S is a convex set, and f is a convex L-smooth function. Frank-Wolfe progresses in two steps first, it approximates the function $f(x)$ locally with a linear function using its First-order Taylor expansion:

$$\min_{x\in\mathbb{R}^P} f(x_t) + \langle f(x_t), x - x_t\rangle$$
$$\text{s.t } x \in S$$

Then to obtain a solution y_t to this problem, we solve the following optimization problem:

$$\min_{x \in \mathbb{R}^P} f(x_t) + \langle f(x_t), x \rangle$$
$$\text{s.t } x \in S$$

Moreover, using the solution y_t one can find the direction of descent $d_t = y_t - x_t$ Hence we derive the following descent iteration:

$$x_{t+1} = x_t + \eta_t d_t$$
$$= (1 - \eta_t)x_t + \eta_t y_t$$
$$\text{s.t } \eta_t = \frac{2}{t+2}$$

Hence the Motivation for using Frank-Wolfe algorithm as a back-bone for many algorithms to obtain a low-rank matrix SDP solution is that if the algorithm is initialized with a matrix of rank 1 and if each minimizer is a matrix of rank 1, then the k^{th} iteration result is the matrix A s.t $rank(A) \leq k$. This is crucial in order to be able to control the rank of the output solution.

3.1 Hazan's Algorithm

The first example of a Frank-Wolfe-based algorithm is Hazan's algorithm.
Hazan's algorithm provides low rank solutions to SDP problems of the form defined in Section 1.1:

$$\min_{x \in S} \langle C, X \rangle$$
$$\text{s.t. } \langle A_i, X \rangle = b_i, i = 1, \ldots, m$$
$$X \succeq 0 \text{ and } Tr(x) \leq t$$

Suppose Q is the set of matrices of the form $\{X \succeq 0, Tr(X) = 1\}$, Hazan's algorithm then tries to optimize this problem:

$$\min_{X \in Q} f(X)$$

For the appropriate choice of function $f(X)$, this optimization is equivalent to finding low-rank solutions to the SDP problem defined earlier in this section [6].

Hazan's Algorithm is defined as following:

Algorithm 2: Hazan's Algorithm

Input: A_1, ..., $A_m \in \mathbb{R}^{nxn}$, $b_1, ..., b_m \in \mathbb{R}$: Set of Constraints, k: The desired maximum rank for the solution,

Output: X: $\frac{1}{k}$-Approximation to the optimization problem s.t $f(X) \leq f^* + \Omega(\frac{1}{k})$ with rank at most k

1 Let M $\leftarrow klog(m)$

2 Initialize X_0 to be an arbitrary matrix $v_0 v_0^T$ with rank 1 ($\mathrm{Tr}(X_0) = 1$)

3 let $f(X) \leftarrow -\frac{1}{M} log(\sum_{i=1}^{m} e^{M(\mathrm{Tr}(A_i, X - b_i))})$

4 **for** $i \leftarrow 1; i \leq k; i = i + 1$ **do**

5 Compute v_i corresponding to the max Eigenvalue of $\nabla f(X_i)$ written in Matrix form

6 set $\eta_i \leftarrow \min(1, \frac{2}{i})$

7 set $X_{i+1} \leftarrow X_i + \eta_i v_i v_i^T - \eta_i X_i$

8 return X_k

Observe that in line 5 in the algorithm, the maximum Eigenvalue and it's corresponding Eigenvector are calculated. However, These are, in fact, approximations rather than exact values, and can by computed in linear time, using the Lanczos algorithm [7] with a random start.

Hazan's algorithm, presented above is guranteed to produce a $\frac{1}{k}$-approximation to the solution of rank at most k, with at most $O(k^2)$ iterations for general SDPs and at most $O(k)$ iterations for optimization over the bounded SDP cone [6] yielding convergence rates of $O(\frac{1}{k^2})$ and $O(\frac{1}{k})$ respectively. It's worth noting that this algorithm applied to general SDPs, produces virtually identical results to the multiplicative weights update algorithm [1], with the same guarantees.

3.2 In-Face Extended Frank-Wolfe

The problem with Hazan's Algorithm is that despite the fact that the rank of the final solution might be low, the ranks of the intermediate matrices don't increase monotonically up to the desired rank before termination, rather they might grow larger than the desired rank in an intermediate step then start decreasing towards the desired rank before termination.

A problem that consequently arises is the Computational and Storage requirements for the high-rank intermediate iteration returns. In this section we introduce the In-Face Extended Frank-Wolfe method which avoids this problem by instead of moving towards the "regular" Frank-Wolfe direction every iteration which might include iterations that result in High-rank matrices, it chooses between the aforementioned "regular FW" direction and moving into a direction that, while still getting it closer to the optimal solution, keeps it "In-Face" meaning in a region - which will be defined below - that keeps the rank of the resulting iteration matrix low. Problems that can be solved using this method are usually of the following type:

$$\min_{x \in \mathbb{R}^{nxd}} f(AX)$$
$$\text{s.t.} ||X||_* \leq 1$$

Such that $f : \mathbb{R}^m \to \mathbb{R}$ is convex and differentiable, $||.||_*$ denotes the Nuclear norm, and $A : \mathbb{R}^{nxd} \to \mathbb{R}^m$ maps X to $(\text{Tr}(A_1X), ..., \text{Tr}(A_mX))^T$. Hence Given this setup we can apply Frank-Wolfe, in particular the extend In-Face Frank-Wolfe.
the initial value X_0 is a rank-1 matrix, and at each iteration i, the algorithm decides between two steps as aforementioned.

First, the regular step which is comprised of computing the direction of descent by finding a pair (v_i, w_i) of singular vectors coresponding to the maximum singular value of $A^*(\nabla f(AX_i))$ s.t $A^* : \mathbb{R}^m \to \mathbb{R}^{nxd}$ is the adjoint operator to A. Then the new iterate is constructed by:

$$X_{i+1} = X_i - \eta X_i - \eta w_i v_i^T$$

Second, the "In-Face step" Which moves into the direction of the minimal face of the nuclear norm unit ball to which X_i belongs to (Denoted $\mathcal{F}(X_i)$). One method to move in the "In-Face" direction is to minimize the following:

$$\begin{aligned} &\text{Tr}((A^*\nabla f(AX_i))^T X) \\ &\text{s.t} X \in \mathcal{F}(X_i) \end{aligned}$$

The resultant matrix from this minimization Y_i produces the next iteration of Extended In-Face Franke-wolfe as following:

$$X_{i+1} = X_i - \eta X_i + \eta Y_i$$

The second, "In-Face" step, is the main idea behind the Extended algorithm, because it keeps the rank of the matrix below a desired k if the choice between Regular and In-Face steps was made appropriately.

Finally, it's shown in [5] a $\frac{1}{k}$-approimation solution can be obtained after $O(k)$ iterations yielding a convergence rate of $O(\frac{1}{k})$, with the rank of the iterate steps always being below k (resolving the storage and computation issues raised earlier by Hazan's algorithm)

4 Discussion

SDP techniques has a wide variety of applications in many fields. In this paper we focused on introducing the current state-of-art techniques in generating low-rank solutions to SDP problems. The first family of methods that we discussed is the Burer-Monteiro methods. This family of methods use low-rank factorization to remove the low-rank positive definite constraint. Within this family of methods, the augmented Lagrangian method uses penalization and Lagrange multipliers to account for constrains of the form $\mathrm{Tr}(A_i V V^\top) = b_i, i = 1, \cdots, m$, while the block coordinate maximization methods account for diagonal constraints using projection for each coordinate. Both classes of methods achieve theoretical and practical convergence. The momentum-accelerated mixing method shows acceleration over existing methods experimentally. For future works, we will try to connect its experimental performance with theory.
The second family of methods that we discussed is the Frank-Wolfe based methods. This family of methods uses Frank-Wolfe algorithm as a back-bone for their computation. The two algorithms presented in this family of methods are Hazan's Algorithm, and Extended In-Face Frank-Wolfe Algorithm. A problem that was noticable with Hazan's algorithm was it's high-rank intermediate iteration returns, which was the main motivation for creating the In-face Frank-Wolfe algorithm's to fix this problem.
Despite low-rank solutions being the main focus of the paper, we also argued that these techniques are still applicable in general; since low-rank solutions are plausible candidates for approximating general SDP solutions. Another possible future direction is to characterize the global minima of SDPs after low-rank factorization and relate them to the global minima of the original problem.

References

[1] S. Arora, E. Hazan, and S. Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*. IEEE, 2005.

[2] Samuel Burer and Renato D.C. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming, Series B*, 95(2):329–357, feb 2003.

[3] Samuel Burer and Renato D.C. Monteiro. Local minima and convergence in low-rank semidefinite programming. *Mathematical Programming*, 103(3):427–444, jul 2005.

[4] Murat A. Erdogdu, Asuman Ozdaglar, Pablo A. Parrilo, and Nuri Denizcan Vanli. Convergence Rate of Block-Coordinate Maximization Burer-Monteiro Method for Solving Large SDPs. jul 2018.

[5] Robert M. Freund, Paul Grigas, and Rahul Mazumder. An extended frank–wolfe method with "in-face" directions, and its application to low-rank matrix completion. *SIAM Journal on Optimization*, 27(1):319–346, jan 2017.

[6] Elad Hazan. Sparse approximate solutions to semidefinite programs. In *Lecture Notes in Computer Science*, pages 306–316. Springer Berlin Heidelberg, 2008.

[7] J. Kuczyński and H. Woźniakowski. Estimating the largest eigenvalue by the power and lanczos algorithms with a random start. *SIAM Journal on Matrix Analysis and Applications*, 13(4):1094–1122, oct 1992.

[8] Anirudha Majumdar, Georgina Hall, and Amir Ali Ahmadi. A Survey of Recent Scalability Improvements for Semidefinite Programming with Applications in Machine Learning, Control, and Robotics. aug 2019.

[9] Po-Wei Wang, Wei-Cheng Chang, and J. Zico Kolter. The Mixing method: low-rank coordinate descent for semidefinite programming with diagonal constraints. *arXiv*, jun 2017.

[10] Alp Yurtsever, Joel A. Tropp, Olivier Fercoq, Madeleine Udell, and Volkan Cevher. Scalable Semidefinite Programming. dec 2019.

Cross-Device Federated Learning: Progress and Open Challenges

Anonymous Authors[1]

Abstract

Federated Learning (FL) is a very hot research topic for a long time. It requires the collaboration between many areas (System, Security/Privacy, Machine Learning). There are two kinds of federated learning: 1. cross-device federated learning 2. cross-silo federated learning. This survey will mainly talk about the researches span across different sub areas of cross device federated learning.

1. Introduction

Federated Learning (FL) is a distributed machine learning approach that can be trained on a large amount of device, like cell-phones and IOT devices. Decentralized data residents on these devices. FL is a more general example "Bring model into the data instead data to model" and solves the underlying problem privacy, ownership, and data locality.

Many efforts have recently been devoted to implementing federated learning algorithms to support effective machine learning models. Specifically, researchers try to support more machine learning models with different privacy-preserving approaches, including deep neural networks (NNs) [(Liu et al., 2020), (Yurochkin et al., 2019), (Bonawitz et al., 2019), (Ryffel et al., 2018), (McMahan et al., 2017)], gradient boosted decision trees (GBDTs) [(Li et al., 2020), (Cheng et al., 2019), (Li et al., 2019)], logistics regression [(Nikolaenko et al., 2013)] and support vector machines (SVMs) [(Smith et al., 2018)]. For instance, Nikolaenko et al. and Chen et al. propose approaches to conduct FL based on linear regression. Hardy et al. [(Hardy et al., 2017)] implement an FL framework to train a logistic regression model. Since GBDTs have become very successful in recent years [(Chen & Guestrin, 2016), (Wen et al., 2020)], the corresponding Federated Learning Systems (FLSs) have also been proposed by [(Zhao et al., 2018)], [(Cheng et al., 2019)], [(Li et al., 2019)]. Another popular ensemble method of decision trees, i.e., random forests, has also been extended to support privacy-preserving [144], which is an important step towards supporting FL. Moreover, there are many neural network based FLSs. Google proposes a scalable production system which enables tens of millions of devices to train a deep neural network. Yurochkin et al. develops a probabilistic FL framework for neural networks by applying Bayesian nonparametric machinery. Several methods try to combine FL with machine learning techniques such as multi-task learning and transfer learning. Smith et al. combine FL with multi-task learning to allow multiple parties to complete separate tasks. To address the scenario where the label information only exists in one party, Yang et al. [197] adopt transfer learning to collaboratively learn a model.

One important design principle of federated learning infrastructure is whether choose asynchronous or synchronous training algorithm. Although there are many successful work on asynchronous federated learning, recently there is one trend on large batch synchronous distributed training. There is somehow close to the setting of distributed training in data-centers.

Apart from the issue of long training time of deep learning models, the conventional training approach requires the entire training dataset has to be stored in the same location. This raises the privacy concern as the data owners do not want their data to leave their premises, especially the data contains sensitive information such as medical records, bank transactions, security logs, etc. This privacy concern prevents the data owners from contributing their data to the training process even though they might know that their data could improve the model performance. This motivates us to develop a novel training framework that allows the model to be trained on different private datasets without relocating/gathering them to the same location. Without trusting any third parties including training coordinator and data owners, such a training approach needs to ensure that there is not any sensitive data leaked, thus preventing a data owner or training coordinator from inferring the data of other owners.

In this report, we will mainly talk about two aspects of federated learning. One is federated aggregation algorithm. The other is the systems for support cross-device/cross-silo federated learning.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

1.1. Algorithms for Federated Learning

Robust Estimation: Robust estimation was pioneered by Huber. Robust median-of-means approaches were introduced. Robust mean estimation, in particular, received much attention. These works consider the statistics of robust estimation in the i.i.d. case, while we focus on distributed optimization with privacy-preservation.

Geometric Median Algorithms: The classical algorithm of Weiszfeld [63] has received much attention [7, 34, 36, 62]. However, all these variants are not numerically stable, while our variant is (cf. Remark 8). A landmark theoretical construction led to a nearly-linear time algorithm for the geometric median [20], but its practical applicability is unclear.

Byzantine Robustness: Byzantine robustness, resilience to arbitrary, even adversarial behavior of some devices [37], has been studied in in gradient based updates [3, 11, 16, 17, 65]. In this work, we consider a more nuanced and less adversarial corruption model because cryptographic protocols which make up secure aggregation require faithful participation of the devices and thus, Byzantine robustness is a priori not possible without additional assumptions. In addition, our setting requires faithful participation of devices in the aggregation loop — see Sec. 2 for examples of its practical relevance. Further, it is unclear how to securely implement the nonlinear aggregation algorithms of these works. Lastly, we note that the use of, e.g., secure enclaves [61] in conjunction with the approach proposed here could guarantee Byzantine robustness in FL.

1.2. Systems for Federated Learning

MapReduce: For datacenter applications, it is now commonly accepted that MapReduce (Dean Ghemawat, 2008) is not the right framework for ML training. For the problem space of FL, MapReduce is a close relative. One can interpret the FL server as the Reducer, and FL devices as Mappers. However, there are also fundamental technical differences compared to a generic MapReduce framework. In our system, FL devices own the data on which they are working. They are fully self-controlled actors which attend and leave computation rounds at will. In turn, the FL server actively scans for available FL devices, and brings only selected subsets of them together for a round of computation. The server needs to work with the fact that many devices drop out during computation, and that availability of FL devices varies drastically over time. These very specific requirements are better dealt with by a domain specific framework than a generic MapReduce.

Distributed ML There has been significant work on distributed machine learning, and large-scale cloud-based systems have been described and are used in practice. Many systems support multiple distribution schemes, including model parallelism and data parallelism, e.g., Dean et al. (2012) and Low et al. (2012). Our system imposes a more structured approach fitting to the domain of mobile devices, which have much lower bandwidth and reliability compared to datacenter nodes. We do not allow for arbitrary distributed computation but rather focus on a synchronous FL protocol. This domain specialization allows us, from the system viewpoint, to optimize for the specific use case. A particularly common approach in the datacenter is the parameter server, e.g., Li et al. (2014); Dean et al. (2012); Abadi et al. (2016), which allows a large number of workers to collaborate on a shared global model, the parameter vector. Focus in that line of work is put on an efficient server architecture for dealing with vectors of the size of 109 to 1012. The parameter server provides global state which workers access and update asynchronously. Our approach inherently cannot work with such a global state, because we require a specific rendezvous between a set of devices and the FL server to perform a synchronous update with Secure Aggregation.

Alternative Approaches Pihur et al. (2018) proposes an algorithm that learns from users' data without performing aggregation on the server and with additional formal privacy guarantees. However, their work focuses on generalized linear models, and argues that their approach is highly scalable due to avoidance of synchronization and not requiring to store updates from devices. Our server design described in Sec. 4, rebuts the concerns about scalability of the synchronous approach we are using, and in particular shows that updates can be processed online as they are received without a need to store them. Alternative proposals for FL algorithms include Smith et al. (2017); Kamp et al. (2018), which would be on the high-level compatible with the system design described here. In addition, Federated Learning has already been proposed in the context of vehicle-to-vehicle communication (Samarakoon et al., 2018) and medical applications (Brisimi et al., 2018). While the system described in this work as a whole does not directly apply to these scenarios, many aspects of it would likely be relevant for production application. Nishio Yonetani (2018) focuses on applying FL in different environmental conditions, namely where the server can reach any subset of heterogeneous devices to initiate a round, but receives updates sequentially due to cellular bandwidth limit. The work offers a resource-aware selection algorithm maximizing the number of participants in a round, which is implementable within our system.

2. Aggregation Algorithms

To protect user privacy, the main responsibility of server is to aggregation all the models uploaded by random selected devices. To achieve the goal towards fairness, privacy, better

device personalization and better accuracy, there are many ways to do aggregation.

2.1. FedAvg

In the federated setting, there is little cost in wall-clock time to involving more clients, and so for our baseline we use large-batch synchronous SGD; experiments by (**?**) [8] show this approach is state-of-the-art in the data center setting, where it outperforms asynchronous approaches. To apply this approach in the federated setting, we select a C fraction of clients on each round, and compute the gradient of the loss over all the data held by these clients. A typical implementation of FedSGD with $C = 1$ and a fix learning rate η has each client k compute $g_k = \nabla F_k(w_t)$, the average gradient on its local data at the current model w_t, and the central server aggregates these gradients and applies the update $w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k$, since $\sum_{k=1}^{K} \frac{n_k}{n} g_k = \nabla f(w_t)$. We term this approach Federated Averaging (FedAvg).

2.2. q-Fair Federated Learning (q-FFL)

Inspired by fair resource allocation in wireless networks, a novel optimization objective is proposed - q-Fair Federated Learning ($q - FFL$). This objective encourages the fair resource allocation across devices in federated learning environment. To help solve the $qFFL$ objective, a communication-efficient method $q - FedAvg$ is proposed.

$$\min_w f_q(w) = \sum_{k=1}^{m} \frac{p_k}{q+1} F_k^{q+1}(w) \tag{1}$$

2.3. Scaffold

Scaffold is a solution uses control variates (variance reduction) to correct for the client-drift in device local updates. SCAFFOLD requires significantly fewer communication rounds and is not affected by data heterogeneity or client sampling. Further, we show that (for quadratics) SCAFFOLD can take advantage of similarity in the client's data yielding even faster convergence. The latter is the first result to quantify the usefulness of local-steps in distributed optimization.

$$\boldsymbol{y}_i \leftarrow \boldsymbol{y}_i - \eta_l \left(g_i(\boldsymbol{y}_i) + \boldsymbol{c} - \boldsymbol{c}_i \right) \tag{2}$$

Here η_l is the local step-size. Then the clients' updates $y_i - x$ are aggregated to form the new server model using a global step-size η_g as:

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + \frac{\eta_g}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} (\boldsymbol{y}_i - \boldsymbol{x}) \tag{3}$$

2.4. FedProx

To tackle both the system heterogeneity and statistical heterogeneity in federated learning, a new federated learning framework is introduced - FedProx. FedProx can be viewed as a generalization and re-parametrization of FedAvg. While this re-parameterization makes only minor modifications to the method itself, these modifications have important ramifications both in theory and in practice. Practically, FedProx allows for more robust convergence than FedAvg across a suite of realistic federated datasets. Here, $||w - w^t||^2$ is the regularizer.

$$\min_w h_k\left(w; w^t\right) = F_k(w) + \frac{\mu}{2} \left\| w - w^t \right\|^2 \tag{4}$$

3. Personalized Federated Learning

In federated learning, each user can solve its local problem defined without any exchange of information with other users; however, the resulted model may not generalize well to new samples as it has been trained over a small number of samples. If users cooperate and exploit the data available at all users, then their local models could obtain stronger generalization guarantees. However, the aggregated model cannot guarantees that it will also achieve good performance on the user local data. So it is very important to incorporates personalization. There are many ways to realize personalization.

3.1. Adaptive Personalized Federated Learning

The study of the degree of personalization in the joint learning algorithm shows that only maximum the performance of the global model limits the ability of the local model to personalize. A adaptively personalized federated learning algorithm is proposed (APFL). In this algorithm, each customer trains them local model, while contributing to the global model. The generalization bound of mixture of local and global models is also derived, and find the optimal mixing parameter. Each device will maintain three models: 1. Global model w_i^t, 2. Local model v_i^t 3. Mixed personalized model $\bar{v}_i^t = \alpha_i v_i^t + (1 - \alpha_i) w_i^t$. During each communication round, it will update all three models following the rules: $\boldsymbol{w}_i^{(t)} = \boldsymbol{w}_i^{(t-1)} - \eta_t \nabla f_i \left(\boldsymbol{w}_i^{(t-1)}; \xi_i^t \right)$, $\boldsymbol{v}_i^{(t)} = \boldsymbol{v}_i^{(t-1)} - \eta_t \nabla_{\boldsymbol{v}} f_i \left(\overline{\boldsymbol{v}}_i^{(t-1)}; \xi_i^t \right)$, $\overline{\boldsymbol{v}}_i^{(t)} = \alpha_i \boldsymbol{v}_i^{(t)} + (1 - \alpha_i) \boldsymbol{w}_i^{(t)}$.

3.2. pFedMe

One challenge associated with FL is statistical diversity among clients, which restricts the global model from delivering good performance on each client's task. To address this, an algorithm is proposed to realize personalized FL (pFedMe) using Moreau envelopes as clients' regularized loss functions, which help decouple personalized model optimization from the global model learning in a bi-level problem stylized for personalized FL. pFedMe updates the global model similarly to the standard FL algorithm such as FedAvg. Similar to FedProx, each local global will also

have a regularizer.

$$\min_{w \in R^d} \left\{ F(w) := \frac{1}{N} \sum_{i=1}^{N} F_i(w) \right\}$$

$$\text{where } F_i(w) = \min_{\theta_i \in R^d} \left\{ f_i\left(\theta_i\right) + \frac{\lambda}{2} \left\| \theta_i - w \right\|^2 \right\}$$

3.3. FL+MAML

Federated Learning wants to train models across multiple computing units (users). This mechanism exploits the computational power of all users and allows users to obtain a richer model as their models are trained over a larger set of data points. However, this scheme only develops a common output for all the users, and, therefore, it does not adapt the model to each user. This is an important missing feature, especially given the heterogeneity of the underlying data distribution for various users. A personalized variant of the federated learning is proposed to help find an initial shared model that current or new users can easily adapt to their local dataset by performing one or a few steps of gradient descent with respect to their own data. This approach keeps all the benefits of the federated learning architecture, and, by structure, leads to a more personalized model fo each user. Inspired by Model-Agnostic Meta-Learning (MAML) framework, a personalized variant of the well-known Federated Averaging algorithm is adapted.

$$\min_{w \in R^d} F(w) := \frac{1}{n} \sum_{i=1}^{n} f_i\left(w - \alpha \nabla f_i(w)\right)$$

4. Backdoor Attack

Federated models are created by aggregating model updates submitted by participants. To protect confidentiality of the training data, the aggregator by designing has no visibility into how these updates are generated. We show that this makes federated learning vulnerable to a model-poisoning attack that is significantly more powerful than poisoning attacks that target only the training data.

4.1. How To Backdoor Federated Learning

4.2. Attack of the Tails

Due to its decentralized nature, Federated Learning (FL) lends itself to adversarial attacks in the form of backdoors during training. The goal of a backdoor is to corrupt the performance of the trained model on specific sub-tasks (e.g., by classifying green cars as frogs). A range of FL backdoor attacks have been introduced in the literature, but also methods to defend against them, and it is currently an open question whether FL systems can be tailored to be robust against backdoors. In general case, robustness to backdoors implies model robustness to adversarial examples, a major open problem in itself. Furthermore, detecting the presence of a backdoor in a FL model is unlikely assuming first order oracles or polynomial time. An edge-case backdoor forces a model to misclassify on seemingly easy inputs that are however unlikely to be part of the training, or test data, i.e., they live on the tail of the input distribution. We explain how these edge-case backdoors can lead to unsavory failures and may have serious repercussions on fairness, and exhibit that with careful tuning at the side of the adversary, one can insert them across a range of machine learning tasks (e.g., image classification, OCR, text prediction, sentiment analysis).

4.3. Distributed Backdoor Attacks

Backdoor attacks aim to manipulate a subset of training data by injecting adversarial triggers such that machine learning models trained on the tampered dataset will make arbitrarily (targeted) incorrect prediction on the testset with the same trigger embedded. While federated learning (FL) is capable of aggregating information provided by different parties for training a better model, its distributed learning methodology and inherently heterogeneous data distribution across parties may bring new vulnerabilities. Distributed backdoor attack (DBA) — a novel threat assessment framework developed by fully exploiting the distributed nature of FL. DBA decomposes a global trigger pattern into separate local patterns and embed them into the training set of different adversarial parties respectively. Compared to standard centralized backdoors, we show that DBA is substantially more persistent and stealthy against FL on diverse datasets such as finance and image data.

5. Semi-supervised Federated Learning

While existing federated learning approaches mostly require that clients have fully-labeled data to train on, in realistic settings, data obtained at the client side often comes without any accompanying labels. Such deficiency of labels may result from either high labeling cost, or difficulty of annotation due to requirement of expert knowledge. Thus the private data at each client may be only partly labeled, or completely unlabeled with labeled data being available only at the server, which leads us to a new problem of Federated Semi-Supervised Learning (FSSL). This new problem of semi-supervised learning under federated learning framework, and propose a novel method to tackle it, which we refer to as Federated Matching (FedMatch). FedMatch improves upon naive federated semi-supervised learning approaches with a new inter-client consistency loss and decomposition of the parameters into parameters for labeled

and unlabeled data. Through extensive experimental validation of our method in two different scenarios, we show that our method outperforms both local semi-supervised learning and baselines which naively combine federated learning with semi-supervised learning.

$$\ell_{\text{final}}(\boldsymbol{\theta}) = \ell_s(\boldsymbol{\theta}) + \ell_u(\boldsymbol{\theta}) \tag{5}$$

6. Other Ideas

6.1. FL with Heterogeneous Architectures

Sharing model updates is typically limited only to homogeneous FL architectures, i.e., the same model is shared with all participants. It would be interesting to study how to extend FL to collaboratively train models with heterogeneous architectures [Gao et al., 2019; Chang et al., 2019], and whether existing attacks and privacy techniques can be adapted to this paradigm

6.2. Decentralized Federated Learning

Decentralized FL where no single server is required in the system is currently being studied [Yang et al., 2019b; Lyu et al., 2019]. This is a potential learning framework for collaboration among businesses which do not trust any third party. In this paradigm, each party could be elected as a server in a round robin manner. It would be interesting to investigate if existing threats on server-based FL still apply in this scenario. Moreover, it may open new attack surfaces. One possible example is that the last party who was elected as the server is more likely to effectively contaminate the whole model if it chooses to insert backdoors. This resembles the fact in server-based FL models which are more vulnerable to backdoors in later rounds of training nearing convergence. Similarly, if decentralized training is conducted in a "ring all reduce" manner, then any malicious participant can steal the training data from its neighbors

References

Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečný, J., Mazzocchi, S., McMahan, H. B., Overveldt, T. V., Petrou, D., Ramage, D., and Roselander, J. Towards federated learning at scale: System design, 2019.

Chen, T. and Guestrin, C. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2016. doi: 10.1145/2939672.2939785. URL http://dx.doi.org/10.1145/2939672.2939785.

Cheng, K., Fan, T., Jin, Y., Liu, Y., Chen, T., and Yang, Q. Secureboost: A lossless federated learning framework, 2019.

Hardy, S., Henecka, W., Ivey-Law, H., Nock, R., Patrini, G., Smith, G., and Thorne, B. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption, 2017.

Li, Q., Wen, Z., and He, B. Practical federated gradient boosting decision trees, 2019.

Li, Q., Wu, Z., Wen, Z., and He, B. Privacy-preserving gradient boosting decision trees. In *AAAI*, 2020.

Liu, Y., Kang, Y., Xing, C., Chen, T., and Yang, Q. A secure federated transfer learning framework. *IEEE Intelligent Systems*, 35(4):70–82, Jul 2020. ISSN 1941-1294. doi: 10.1109/mis.2020.2988525. URL http://dx.doi.org/10.1109/MIS.2020.2988525.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data, 2017.

Nikolaenko, V., Weinsberg, U., Ioannidis, S., Joye, M., Boneh, D., and Taft, N. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, pp. 334–348, 2013. doi: 10.1109/SP.2013.30.

Ryffel, T., Trask, A., Dahl, M., Wagner, B., Mancuso, J., Rueckert, D., and Passerat-Palmbach, J. A generic framework for privacy preserving deep learning, 2018.

Smith, V., Chiang, C.-K., Sanjabi, M., and Talwalkar, A. Federated multi-task learning, 2018.

Wen, Z., Liu, H., Shi, J., Li, Q., He, B., and Chen, J. Thundergbm: Fast gbdts and random forests on gpus. *Journal of Machine Learning Research*, 21(108):1–5, 2020. URL http://jmlr.org/papers/v21/19-095.html.

Yurochkin, M., Agarwal, M., Ghosh, S., Greenewald, K., Hoang, T. N., and Khazaeni, Y. Bayesian nonparametric federated learning of neural networks, 2019.

Zhao, L., Ni, L., Hu, S., Chen, Y., Zhou, P., Xiao, F., and Wu, L. Inprivate digging: Enabling tree-based distributed data mining with differential privacy. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 2087–2095, 2018. doi: 10.1109/INFOCOM.2018.8486352.