CS230

Challenges in Scalable Distributed Training of Deep Neural Networks under communication constraints (Project Report for CS230, Winter 2021)

Kateryna Pistunova
kpistunova@stanford.edu

Rajarshi Saha
rajsaha@stanford.edu

Akshay Nalla
analla@stanford.edu

1 Introduction

In the present era of ever-growing datasets and model sizes, distributed training of neural networks has become a crucial paradigm (1), (2). In this project, we will explore the challenges that come up when multiple machines train a model together in a decentralized fashion. When multiple nodes collaborate to train a model together over a network, practical constraints such as *network bandwidth* and *latency* overwhelm the expected benefits of the distributed framework.

We consider the problem of decentralized non-convex optimization under communication constraints, and propose an algorithm: **Subsampled Decentralized Stochastic Gradient Descent** (*Subsampled - DSGD*), an algorithm in which a node randomly subsamples a vector and projects it onto a lower dimensional subspace before communicating with other nodes, thereby reducing communication demands. We give our algorithm pseudocode in Algo. 2.

2 Related Work

In decentralized learning, the objective is to solve an optimization problem in a fully distributed setting by keeping the data on the nodes/edge-devices themselves, and there is no central server orchestrating the operation of the individual nodes. Models for decentralized information exchange and computation were first proposed by Tsitsiklis et al. (3) (4). Nedić and Ozdagler (5) built upon their model and proposed subgradient methods for functions distributed across multiple agents. Duchi et al. (6) proposed distributed version of primal-dual methods (7) and studied the dependence of convergence rates of the algorithm on network topology. Tsianos et al. (8) used strongly convex assumption of the objective functions to propose an algorithm for online convex optimization problems.

In recent years, distributed learning makes it possible to greatly accelerate the training of modestly sized models, and also to train models with humongous amounts of data which couldn't be contemplated otherwise (9). Works like (10), (11), (12) propose techniques like increasing edge computation and intermittently transmitting gradient updates to the central server, compression schemes like sparsification, subsampling, and quantization of the message transmitted in order to reduce communication demands of distributed training. However all of these works consider the parameter-server framework in which a central server is responsible for periodically aggregating the local updates, taking a consensus, and broadcasting the updated model back to the edge devices. In order words, this setup is not completely "decentralized".

Decentralized settings are attractive because they are philosophically more democratic. They are often preferred due to privacy concerns, making the system more robust to node failures, and might be the only option in ad-hoc settings in the absence of a parameter-server infrastructure. Moreover, it has been shown that decentralized training can be faster than distributed training with a central server when operating on networks with low bandwidth or high latency (13).

Decentralized algorithms for non-convex optimization problems are considered in (14), (15), (16). They show convergence of decentralized SGD to a stationary point of the non-convex objective function. Our work is similar to the communication compression schemes proposed in (17). However, we note that the sketching scheme used in *Subsampled-DSGD* for dimension-

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

Algorithm 1 Subsampled-DSGD algorithm

Input: Max. iters. K, Subsampling ratio $\gamma = \frac{d'}{d}$,
Initial values $\theta_0^{(i)}$, for $i \in \{1, \ldots, n\}$
Output: Primal iterates $\theta_K^{(i)}$, for $i \in \{1, \ldots, n\}$

1: **while** $k < K$ **do** node i update as:
2: Node $j \in \mathcal{N}(i)$ compresses its iterate as:

$$\tilde{\theta}_k^{(j)} = U_k \theta_k^{(j)}$$

3: Node i receives $\tilde{\theta}_k^{(j)}$ from all $j \in \mathcal{N}(i)$.
4: Node i reconstructs its neighbors' states as:

$$\hat{\theta}_k^{(j)} = \frac{1}{\gamma} U_k^H \tilde{\theta}_k^{(j)}$$

5: Node i does local computation to get a stochastic gradient $g(\theta_k^{(i)})$, and update its own iterate as:

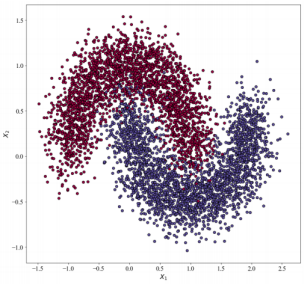
$$\theta_{k+1}^{(i)} = \sum_{j \in \mathcal{N}(i)} P_{ij} \hat{\theta}_k^{(j)} - \eta g(\theta_k^{(i)})$$

6: **end while**
7: **return** $\theta_K^{(i)}$ for $i \in \{1, \ldots, n\}$

ality reduction for model-exchange can be used on top of other schemes like quantization proposed in (17) to further reduce communication demands on the network.

3 Dataset

As part of the current work we have used a synthetic moon-shaped binary classification dataset as visualized on the right.



4 Methods

We propose **Subsampled-DSGD**: an algorithm which reduces communication demands on the network bandwidth and provides communication versus convergence tradeoff. The algorithm pseudocode is given in Algo. 2.

This section presents Subsampled Decentralized Stochastic Gradient Descent (Subsampled-DSGD) as an algorithm which reduces communication demands on the network bandwidth, and provides communication versus convergence tradeoff.

4.1 Sub-sampling model

Consider a unitary matrix $U \in \mathbb{C}^{d \times d}$, i.e. $U^H U = U U^H = I$. Here, $(\cdot)^H$ denotes transposed conjugate operation. Possible choices include identity matrix, Fourier matrix, or a random unitary matrix. At any iteration k, when node i needs to update its iterate, node j samples d' rows of this unitary matrix uniformly at random to get $U_k \in \mathbb{C}^{d' \times d}$. Node j computes the subsampled vector:

$$\tilde{\theta}_k^{(j)} = U_k \theta_k^{(j)} \in \mathbb{C}^{d'}.$$

The vector of reduced dimensionality $\tilde{\theta}_k^{(j)}$, is transmitted over the $(i,j)^{th}$ edge. Node i receives $\tilde{\theta}_k^{(j)}$, and it reconstructs an estimate of node j's iterate as:

$$\hat{\theta}_k^{(j)} = \frac{1}{\gamma} U_k^H \tilde{\theta}_k^{(j)} \in \mathbb{C}^d.$$

where $\gamma = d'/d$ denotes the *subsampling* ratio. Node i then uses this node j's reconstructed iterate to update its own state as:

$$\theta_{k+1}^{(i)} = \sum_{j \in \mathcal{N}(i)} P_{ij} \hat{\theta}_k^{(j)} - \eta g(\theta_k^{(i)}). \tag{1}$$

Algorithm 2 Subsampled-DSGD algorithm

Input: Max. iters. K, Subsampling ratio $\gamma = \frac{d'}{d}$,
Initial values $\theta_0^{(i)}$, for $i \in \{1, \ldots, n\}$

Output: Primal iterates $\theta_K^{(i)}$, for $i \in \{1, \ldots, n\}$

1: **while** $k < K$ **do** node i update as:

2: Node $j \in \mathcal{N}(i)$ compresses its iterate as:

$$\tilde{\theta}_k^{(j)} = U_k \theta_k^{(j)}$$

3: Node i receives $\tilde{\theta}_k^{(j)}$ from all $j \in \mathcal{N}(i)$.

4: Node i reconstructs its neighbors' states as:

$$\hat{\theta}_k^{(j)} = \frac{1}{\gamma} U_k^H \tilde{\theta}_k^{(j)}$$

5: Node i does local computation to get a stochastic gradient $g(\theta_k^{(i)})$, and update its own iterate as:

$$\theta_{k+1}^{(i)} = \sum_{j \in \mathcal{N}(i)} P_{ij} \hat{\theta}_k^{(j)} - \eta g(\theta_k^{(i)})$$

6: **end while**

7: **return** $\theta_K^{(i)}$ for $i \in \{1, \ldots, n\}$

The algorithm pseudocode is given in (2). Note that in the reconstructed estimate $\hat{\theta}_k^{(j)}$, entries corresponding to the rows of the unitary matrix U that were not selected are zero and the remaining entries are as it is except for scaling by a factor of $1/\gamma$. In a sense, this is similar to some of previous sparsification schemes proposed in (10), (18) in the context of parameter-server framework. Note however, that our random subsampling scheme is different from sparsification schemes which retain few elements of maximum magnitude and make the rest zero. Retaining maximum magnitude elements only at every iteration introduces a bias in the estimates which affects the convergence of the algorithm. The random subsampling scheme, on the other hand ensures that the reconstructed iterate $\hat{\theta}_k^{(j)}$ is an unbiased estimate of $\theta_k^{(j)}$.

5 Experiments/Results/Discussion

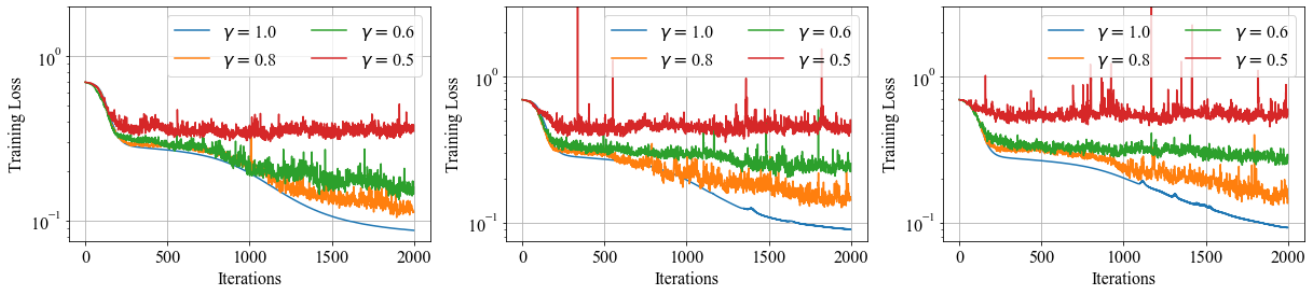
We have evaluated Subsampled-DSGD on the binary classification task for the Moon-shape dataset and the nonlinear regression task for residential energy consumption dataset, and the plots are provided in Figs. 1a, 1b and 1c, which correspond to the training loss performances of the model trained at any particular node, when trained over fully-connected, and ring networks with 2 and 4 neighbors respectively.

The training curves for fully connected and ring networks with various subsampling ratios are shown in Fig. 1. It can be observed that as the subsampling ratio decreases, the training loss plateaus at a higher level, which indicates the degradation of the learning performance due to the communication constraint. For the binary classifcation of moon-shaped dataset, a four-layer fully connected neural network with layer sizes $[2, 25, 25, 1]$ is used as the classifier. The NN architecture can be seen in Fig. 2 The neural network uses ReLU activation in the first three layers, and sigmoid in the output layer. Cross entropy is adopted as the loss function. Each node has a local dataset of 1000 synthetic training samples.

An addition point of interest was to observe how the number of bits being transferred affected the final loss. Fig. 3 shows how the final loss decreases as the bits transferred increase. A point of interest is how the figure shows that when bit transfer is low, the architectures at utilize a greater amount of nodes perform better.

6 Conclusion/Future Work

The next goal in our project is trying to implement the distributed learning algorithm in a more complex deep learning architecture. The goal is to develop a ResNet model that works with the distributed stochastic gradient descent algorithm. The model will be used for a classification task to categorize input images into one of the 10 classes that are specified in our dataset. With the implementation of a working ResNet model, our metric of interest would be the performance of our algorithm versus the communication cost. More specifically, we would be interested in the final training accuracy our model is able to achieve plotted against the amount of data that was exchanged. Our final step would be to identify an open source distributed learning method to compare with our algorithms performance. For example, we can use the results of researchers from Sony (19) where they use distributed stochastic gradient descent for ImageNet/ResNet-50 training with the best training time of as little as 122 seconds and



(a) (b) (c)

Figure 1: Training curves for moon-shape classification task: (a) fully connected node network; (b) ring network with 4 neighbors per node; (c) ring network with 2 neighbors per node.

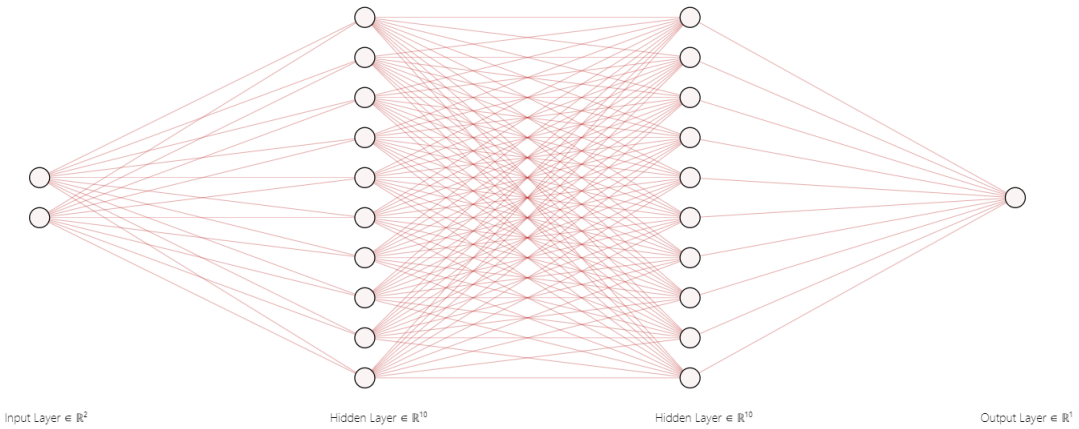


Figure 2: Neural Network Architecture (For clarity, hidden layer depth is set to 10 in the image but it was 25 in the model)

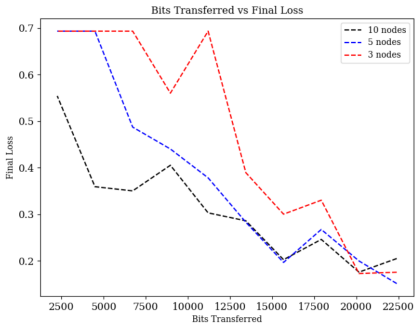


Figure 3: Bits vs final loss for models with various amounts of nodes

validation accuracy of 75.29%. Another way to compare our results is to use the following Github repository (20) where the authors train a distributed ResNet model in Tensorflow using SyncReplicasOptimizer.

7 Contributions

Rajarshi: Worked on the theoretical analyis of the algorithm, and worked on obtaining the training loss vs. iteration curves.

Kateryna: Did literature review, worked on the code, tried to include quantization in the algorithm.

Akshay: Worked on the code base, literature review, and bit versus final loss analysis

8 Acknowledgements

Part of this work was done as a course project for EE364B (Spring 2020) in collaboration with Yun Liao. The authors would also like to thank Nancy Xu and Jonathan Li for their initial guidance on this project.

References

[1] Ligeng Zhu, Yao Lu, Yujun Lin, and Song Han, "Distributed training across the world," 2020.

[2] Max Ryabinin and Anton Gusev, "Towards crowdsourced training of large neural networks using decentralized mixture-of-experts," 2020.

[3] J. N. Tsitsiklis, *Problems in decentralized decision making and computation*, Massachusetts Institute of Technology (MIT), Cambridge, 1984.

[4] J. N. Tsitsiklis, D. Bertsekas, and Athans M., "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE Transactions on Automatic Control*, vol. 31, no. 9, pp. 803–812, Sep 1986.

[5] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, Jan 2009.

[6] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *IEEE Transactions on Automatic Control*, vol. 57, no. 3, pp. 592–606, Mar 2012.

[7] Y. Nesterov, "Primal-dual subgradient methods for convex problems," *Mathematical Programming A*, vol. 120, no. 1, pp. 221–259, June 2009.

[8] K. I. Tsianos and M. G. Rabbat, "Distributed strongly convex optimization," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Oct 2012, pp. 593–600.

[9] Jeffrey Dean and et al, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25*, pp. 1223–1231. 2012.

[10] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon, "Federated learning: Strategies for improving communication efficiency," 2016.

[11] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016.

[12] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," 2017.

[13] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Advances in Neural Information Processing Systems 30*, pp. 5330–5340. 2017.

[14] J. Zeng and W. Yin, "On nonconvex decentralized gradient descent," *IEEE Transactions on Signal Processing*, vol. 66, no. 11, pp. 2834–2848, 2018.

[15] Jianyu Wang and Gauri Joshi, "Cooperative SGD: A unified framework for the design and analysis of communication-efficient sgd algorithms," 2018.

[16] Eduard Gorbunov, Filip Hanzely, and Peter Richtárik, "A unified theory of sgd: Variance reduction, sampling, quantization and coordinate descent," 2019.

[17] Hanlin Tang, Shaoduo Gan, Ce Zhang, Tong Zhang, and Ji Liu, "Communication compression for decentralized training," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, Dec 2018, p. 7663–7673.

[18] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang, "Gradient sparsification for communication-efficient distributed optimization," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, p. 1306–1316.

[19] Hiroaki Mikami, Hisahiro Suganuma, Pongsakorn U-chupala, Yoshiki Tanaka, and Yuichi Kageyama, "Massively distributed sgd: Imagenet/resnet-50 training in a flash," 2019.

[20] Jiarui Fang, "Distributed resnet tensorflow," `https://github.com/feifeibear/Distributed-ResNet-Tensorflow`, 2018.

[21] Léon Bottou, Frank E. Curtis, and Jorge Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

Appendices

A Convergence analysis of Subsampled-DSGD

In the subsequent sections, we present a simple but crude convergence result for *Subsampled-DSGD*. Most of the proof makes simple modifications to the proof of cooperative SGD in (15). To keep the report succint, several proof details are skipped and only the main results are mentioned. The reader is referred to (15) for a comprehensive exposition of the convergence proof.

A.1 Assumptions for convergence analysis

The following assumptions are made for the convergence analysis, on the lines of existing works (15), (21).

1. *Function smoothness*: The objective function satisfies $\|\nabla f(\theta_1) - \nabla f(\theta_2)\|_2 \leq L \|\theta_1 - \theta_2\|_2$ for all $\theta_1, \theta_2 \in \Theta$.
2. *Finite infimum*: $f(\theta) \geq f_{inf} \geq -\infty$.
3. *Compact feasible set*: For all $\theta \in \Theta$, $\|\theta\|_\infty \leq R$ for some finite radius R.
4. *Unbiased gradients*: The stochastic gradient oracle satisfies $\mathbb{E}[g(\theta)] = \nabla f(\theta)$.
5. *Bounded variance*: For some non-negative constant σ, $\mathbb{E}[\|g(\theta) - \nabla f(\theta)\|_2^2] \leq \sigma^2$.
6. *Network matrix*: $\mathcal{G}$ is connected and undirected, and the network is associated with an $n \times n$ doubly stochastic consensus matrix P i.e. $\sum_{i=1}^n P_{ij} = \sum_{j=1}^n P_{ij} = 1$, which respects the structure of $\mathcal{G}$ in the sense that $[P]_{ji} = 0$ iff $(i,j) \notin E$.

Furthermore, since the objective function is non-convex in general, *Subsampled-SGD* is expected to convergence to a stationary point or a saddle point where the gradient is zero. A point θ^* is said to be ϵ-*stationary* if $\|\nabla f(\theta^*)\|_2 < \epsilon$.

A.2 Convergence analysis

This section shows the main result regarding the convergence of *Subsampled-DSGD*. As far as possible, notations and terminology are kept consistent with (15) except when it starts blurring clarity.

The primal iterates of nodes $i = 1, \ldots, n$ at iteration k are denoted as $x_k^{(1)}, \ldots, x_k^{(n)} \in \mathbb{R}^d$. Denote the average of the primal iterates across all the nodes by $u_k \in \mathbb{R}^d$, i.e.

$$u_k = \frac{1}{n} \sum_{i=1}^n x_k^{(i)}.$$

The algorithm is said to achieve an ϵ-*suboptimal* solution after K iterations if the following holds true:

$$\mathbb{E}\left[\frac{1}{K} \sum_{k=1}^K \|\nabla f(u_k)\|_2^2\right] \leq \epsilon.$$

This condition guarantees convergence of the algorithm to an ϵ-*stationary* point. We next state a property of the proposed random subsampling scheme.

Proposition 1. *For any unitary matrix $U \in \mathbb{C}^{d \times d}$, if the rows of the matrix are selected uniformly at random to get $U_k \in \mathbb{C}^{d' \times d}$, and $\hat{\theta} = \frac{1}{\gamma} U_k^H U_k \theta$, then $\hat{\theta}$ is an unbiased estimator of θ with bounded variance, i.e.*

$$\mathbb{E}\left[\hat{\theta}\right] = \theta \;\; \textit{and} \;\; \mathbb{E}\left[\left\|\hat{\theta} - \theta\right\|_2^2\right] \leq \frac{dR^2}{\gamma}.$$

Proof. The proof follows directly from the the observation that to get $\hat{\theta}$ from θ, we select d' out d entries of the vector θ uniformly at random and retain them after scaling by a factor of $1/\gamma$, and making the remaining $d - d'$ entries zero. d' elements can be sampled from d elements in $\binom{d}{d'}$ ways. Suppose θ_j denotes the j^{th} element of $\theta \in \mathbb{R}^d$. Then,

$$\textbf{Prob}\left[\hat{\theta}_j = \frac{1}{\gamma}\theta_j\right] = \frac{\binom{d-1}{d'-1}}{\binom{d}{d'}} = \gamma.$$

So, $\mathbb{E}\left[\hat{\theta}\right]_j = \gamma \frac{1}{\gamma}\theta_j + (1-\gamma) \cdot 0 = \theta_j$, and hence random subsampling gives an unbiased reconstructed estimate. To show the bounded variance part, define $r := \hat{\theta} - \theta$. For a given θ, the random subsampling scheme selects one out of $\binom{d}{d'}$ possibilities for r, uniformly with a

probability $1/\binom{d}{d'}$. So,

$$\mathbb{E}\left[\|r\|_2^2\right] = \frac{1}{\gamma^2}\frac{1}{\binom{d}{d'}}\sum_{\{i_1,\ldots,i_{d'}\}\in[d]}\left(x_{i_1}^2+\ldots+x_{i_{d'}}^2\right)$$
$$\leq \frac{1}{\gamma^2}\frac{1}{\binom{d}{d'}}\binom{d}{d'}\gamma dR^2 = \frac{dR^2}{\gamma}.$$

□

It is worthwhile to note that as the subsampling ratio γ is decreased, the variance of the estimate increases in an inverse proportion.

The analysis that follows is done considering constant step size η. As is the case for subgradient algorithms, with constant step size, the algorithm converges only to a neighborhood of the optimal solution. While training neural networks, step-size is often decreased in an epochal fashion. Constant step size is used to train for an epoch until the model saturates and reaches a sub-optimal neighborhood. In the next epoch, the step-size is reduced (for example, by a factor of $\frac{1}{2}$) and the algorithm is warm-started from the solution obtained from the last epoch, and this process is repeated over multiple epochs to eventually each the minima. The next proposition characterizes the convergence of *Subsampled-DSGD* and shows how the error floor depends on the subsampling ratio.

Proposition 2. *Suppose Subsampled-DSGD is run for K iterations over n nodes. Under assumptions 1-6, if the learning rate η satisfies*

$$\eta L + 5\frac{\eta^2 L^2}{(1-\zeta)^2} \leq 1,$$

where $\zeta = |\lambda_2(P)|$ is the second largest eigenvalue of the network consensus matrix, and all local models are initialized at a same point u_1, then the squared gradient norm averaged across all the nodes after K iterations is bounded as:

$$\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K}\|\nabla f(u_k)\|_2^2\right] \leq \frac{2\left[f(u_k)-f_{inf}\right]}{\eta K} + \frac{\eta L}{n}\left(\frac{dR^2}{\gamma}+\sigma^2\right) + 2\frac{\eta^2 L^2}{1-\zeta^2}\left(\frac{dR^2}{\gamma}+\sigma^2\right).$$

Proof. A very crude proof is presented here. It is essentially a simple extension of Theorem 1 in (15). The critical observation lies in the fact that the stochasticity introduced due to random subsampling can be treated equivalently as that due to the stochastic gradient oracle. Note that from assumptions 1-6, $g(\theta_k^{(i)})$ can be written as $g(\theta_k^{(i)}) = \nabla f(\theta_k^{(i)}) + \nu$, where ν is a zero-mean random variable with variance $\mathbb{E}[\nu^2] \leq \sigma^2$. Moreover, as a consequence of proposition (1), $\hat{\theta}_k^{(j)} = \theta_k^{(j)} + \upsilon$, where υ is zero-mean and has variance $\mathbb{E}[\upsilon^2] \leq \frac{dR^2}{\gamma}$.

Equation (1) can then be written as:

$$\theta_{k+1}^{(i)} = \sum_{j=1}^{n} P_{ij}\hat{\theta}_k^{(j)} + g(\theta_k^{(i)})$$
$$= \sum_{j=1}^{n} P_{ij}\theta_k^{(j)} + \sum_{j=1}^{n} P_{ij}\upsilon_k^{(j)} + \nabla f(\theta_k^i) + \nu_k^{(i)}.$$

Consider the random quantity $\xi_k^{(i)} = \sum_{j=1}^{n} P_{ij}\upsilon_k^{(j)} + \nu_k^{(i)}$. The fact that subsampling is done randomly and independently at each node, the random quantities can be clubbed together and $\xi_k^{(i)}$ can be treated as the effective stochasticity introduced due to the stochastic gradient oracle. Note $\mathbb{E}\left[\xi_k^{(i)}\right] = 0$, and

$$\mathbb{E}\left[{\xi_k^{(i)}}^2\right] \leq \sum_{j=1}^{n} P_{ij}\frac{dR^2}{\gamma} + \sigma^2 = \frac{dR^2}{\gamma} + \sigma^2.$$

The last step utilizes the fact that the network consensus matrix P is doubly stochastic, i.e. $\sum_{j=1}^{n} P_{ij} = 1$. The rest of the proof is similar to the proof of theorem 1 in (15) with σ^2 replaced by $\frac{dR^2}{\gamma} + \sigma^2$. □

For a fixed step-size η, the error floor can be obtained by letting $K \to \infty$ in proposition (2), as follows:

$$\epsilon_{floor} = \frac{\eta L}{n}\left(\frac{dR^2}{\gamma}+\sigma^2\right) + 2\frac{\eta^2 L^2}{1-\zeta^2}\left(\frac{dR^2}{\gamma}+\sigma^2\right). \tag{2}$$

It is evident from (2) that as the subsampling ratio γ is decreased, the error floor goes up, which is to be expected as the price being paid for reduced communication costs. If a cost structure is imposed on the network communication as a function of the subsampling ratio γ, this gives us a design tradeoff between communication costs and the performance of the algorithm, and an appropriate point can be chosen on the Pareto curve depending on system design specifications.