Decentralized Optimization for Machine Learning: Distributed Stochastic Gradient Tracking

Brian Zhang

April 29, 2024

Abstract

Applying machine learning to massively distributed networks, such as sensor or drone arrays, has gained much traction in recent years. Due to the decentralized nature of these agents, it is necessary to develop algorithms that can operate efficiently across massively distributed environments. There are many unique challenges that distributed optimization faces, such as data heterogeneity, privacy, and network robustness. This project explores a contemporary solution to addressing the problems of distributed optimization, called Distributed Stochastic Gradient Tracking (DSGT).

1 Introduction

In recent years, wireless and Internet technology advancements have allowed machine learning to train across massively distributed agent networks rather than rely on a centralized computing source. This has led to myriad machine learning applications, such as robotic networks, autonomous vehicle systems, social networks, and healthcare applications [5]. With these new implementations come unique challenges, a few of which are listed below:

1. **Data heterogeneity among agents:** it cannot be assumed that every agent in a network has the same data distribution to train on. For example, consider a machine learning model for lung cancer detection trained across different hospitals in various regions. Perhaps a particular region has a higher population of young smokers and thus correlates to a younger population with lung cancer. This data distribution would differ in another region with very few young smokers. A distributed optimization framework must account for data heterogeneity to produce generalizable results.

2. **Privacy-sensitive data:** individual agents often cannot freely exchange private data to create a shared data model. For example, in the United States, strict laws (HIPAA) prevent healthcare providers from disclosing sensitive patient data. This means that agents in distributed models must find a way to safely encode their data to a representation that allows for efficient communication of gradient information for every agent to converge to the same solution.

3. **Network topology:** in massively distributed environments, connections between agents in a network are not necessarily constant. For instance, machine learning models for facial recognition, trained on mobile phones, are limited by individual connections to the Internet. Additionally, there may not be a centralized server that can coordinate the communication between each agent. It is necessary to build optimization frameworks that are robust to changes in network structure and resilient to individual agents' failure.

4. **Limited computing resources:** unlike in most centralized optimization settings, agent computing power can be minimal. In particular, remote edge networks, such as sensor arrays in harsh environments, have limited computational power and storage ability [5]. Distributed frameworks seek extreme efficiency to allow for computing even with few resources available to an agent.

There are many different recent developments in the distributed optimization space that have attempted to address these problems. One popular framework that has attempted to address some of these issues, particularly the problems of data privacy and network topology, is Distributed Stochastic Gradient Tracking, or **DSGT**.

2 Problem Formulation

A distributed optimization problem can be described as follows:

$$\begin{aligned} \min_{x_1,x_2,\ldots,x_m \in C} \quad & f(x) = \sum_{i=1}^{m} f_i(x_i) \\ \text{subject to} \quad & x_i = x_j \qquad \forall\, i,j \in \{1,2,\ldots m\} \end{aligned} \tag{1}$$

in a network of m agents, where the constraint set $C \subseteq \mathbb{R}^n$, is known among all agents, but the local objective function f_i is private to the ith agent.

The solution vector x^* that minimizes f is called the *global solution*, and the value $f(x^*) = p^*$ is called the *global optimum*. Likewise, the solution vector x_i^* that minimizes f_i is called the *local solution* of the ith agent, and the value $f_i(x_i^*)$ is the *local optimum* of the ith agent. This is not to be confused with the definitions of local and global optima of an particular function.

An agent network can be represented as a graph $G(V, E)$, where V denotes the set of vertices of the graph ($|V| = m$), and $E = \{(v_1, v_2) \mid v_1 \text{ and } v_2 \text{ are connected}, v_1, v_2 \in V\}$ denotes the set of edges of the graph. From this general setup, some assumptions must be made to make the problem relevant for a convex optimization setting:

- The global constraint set C is closed and convex. Also, the ith agent's local objective function $f_i : \mathbb{R}^n \to \mathbb{R}$ is strongly convex and at least once-differentiable for all agents i. This means that the global objective f is strongly convex as well, and has a unique optimum.
- The agent network graph G is connected, i.e. there exists a path from vertex u to vertex v for every $u, v \in V$. For simplicity, it is also assumed that G is undirected and not time-varying.

An important subproblem in every distributed optimization problem is the *consensus problem*. This refers to the challenge of achieving agreement among network agents on a common solution vector, i.e. the constraint

$$x_1 = x_2 = \cdots = x_m$$

Achieving consensus is crucial for ensuring that all agents collectively minimize the global objective function while converging to the same solution vector [5].

For example, consider the naive approach of each agent i locally running gradient descent to solve f_i. Using this approach, each agent will have their own solution vector x_i^* that minimizes f_i, but there is no guarantee that $x_i^* = x_j^*$ for $i \neq j$.

The consensus problem explicitly written in (1) can be written compactly as an implicitly-constrained distributed problem:

$$\min_{x \in C} \qquad f(x) = \sum_{i=1}^{m} f_i(x) \tag{2}$$

where $x_1 = x_2 = \cdots = x_m = x$. For simplicity, this report explores only the case where $C = \mathbb{R}^n$. That is, the only constraint that the solution must meet is the consensus constraint; there are no individual constraints placed on each agent.

3 ML Optimization & Stochastic Gradient Descent

The distributed algorithms covered in this report are primarily used in machine learning applications. Machine learning optimization takes the special form of optimizing a *loss function* $\mathcal{L}$ across a dataset:

$$\min_x \qquad F(x) = \frac{1}{n} \sum_{d \in \mathcal{D}} \mathcal{L}(d; x) \tag{3}$$

Where $\mathcal{D}$ is a dataset of size $|\mathcal{D}| = n$, d is a point in $\mathcal{D}$, and $\mathcal{L}(d; x)$ is an evaluation of the loss for a given point d and solution vector (model parameter) x. This problem can be thought of as minimizing the average loss over every data point $d \in \mathcal{D}$ for a given x.

For many machine learning problems, the dataset may be extremely large, and thus calculating the gradient over the full dataset can be prohibitively expensive. Moreover, in large datasets, many data points may be extremely similar, and so computing the gradient over all data points can lead to many redundant computations. For this reason, the **Stochastic Gradient Descent** (SGD) algorithm is extremely popular for machine learning applications [7]. DSGT is simply an adaptation of SGD to distributed optimization problems.

Algorithm 1 (Minibatch) Stochastic Gradient Descent

1: **Input:** a random vector x_0, initial diminishing stepsize $\gamma_0 > 0$, and batch size $b > 0$
2: **for** $k = 0, 1, \ldots$ **do**
3: $\quad d^{(i:i+b)} \leftarrow$ a realization of b data points in $\mathcal{D}$
4: $\quad x_{k+1} \leftarrow x_k - \gamma_k \nabla \mathcal{L}(d^{(i:i+m)}; x)$
5: **end for**

The idea of SGD is to take a small sample of the larger dataset and use it as an estimator of the true gradient. Note that if $b = |\mathcal{D}|$, then SGD reduces to gradient descent. Two important assumptions of the stochastic gradient are required for SGD to work:

Lemma 1. *The stochastic gradient $\nabla \mathcal{L}(d; x)$ is an unbiased estimator of the true gradient $\nabla F(x)$:*

$$\mathbb{E}\left[\nabla \mathcal{L}(d; x) \mid x\right] = \nabla F(x)$$

Lemma 2. *The stochastic gradient $\nabla \mathcal{L}(d; x)$ has bounded variance:*

$$\mathbb{E}\left[\left\| \nabla \mathcal{L}(d; x) - \nabla F(x) \right\|^2 \;\middle|\; x \right] \leq \sigma^2 \qquad \text{for all } x \in \mathbb{R}^n, \text{ for some } \sigma > 0$$

These two lemmas together suggest that, for any particular noisy gradient $\nabla \mathcal{L}$, the estimate is within some error of the true gradient ∇F. The noisy gradient is particularly useful for nonconvex problems (i.e. most ML problems), which can allow for the updated vector to jump to different local minima. However, the noisiness of the stochastic gradient also makes it more difficult to reach convergence, as SGD will constantly overshoot a local minimum.

To partially alleviate this issue, *batching* is introduced to decrease the variance of the sampled gradient [7]. Minibatch SGD takes the mean gradient over a small sample of data size b. Since

variance is bounded on each individual term, this means that overall variance is scaled by a factor $\sim 1/b^2$. Batching does not entirely solve the problem of overshooting, so a *diminishing step size* is also necessary to asymptotically decrease the change in x to 0 [7].

Why is this important for distributed optimization? In distributed machine learning, computational agents collectively solve the problem:

$$\min_x \quad f(x) = \frac{1}{n}\sum_{k=1}^{m} n_k F_k(x) \quad \text{where} \quad F_k(x) = \frac{1}{n_k}\sum_{d\in\mathcal{D}_k} \mathcal{L}(d;x) \tag{4}$$

where the full dataset $\mathcal{D}$ is partitioned among m agents, with dataset $\mathcal{D}_k$ belonging to the kth agent, and $|\mathcal{D}_k| = n_k$. Plugging in each agent's individual objective function to the global objective yields

$$\min_x \quad f(x) = \frac{1}{n}\sum_{d\in\mathcal{D}} \mathcal{L}(d;x)$$

Which is the same centralized machine learning problem in (3). This implies that the "full" gradient evaluation of the kth agent, $\nabla F_k(x)$, is actually still a noisy gradient approximation of $\nabla f(x)$. Therefore, performing gradient descent on agent k is equivalent to performing stochastic gradient descent with batch $b = \mathcal{D}_k$, not gradient descent on the full dataset. This means any optimization method for distributed learning inherently relies on SGD as an optimizer.

4 Centralized Consensus and Federated Averaging

Federated Averaging, also known as Federated Learning, is a privacy-sensitive, distributed machine framework pioneered by researchers at Google in 2016 [3]. As its background suggests, this framework was initially designed to allow for practical training of deep neural networks over distributed agents. However, it is also applicable to other optimization problems as well. The term "federated" refers to the loose "federation" of participating devices (agents) that work together to solve an optimization problem. Federated Averaging solves the distributed machine learning problem described in (4):

$$\min_x \quad f(x) = \frac{1}{n}\sum_{k=1}^{m} n_k F_k(x) \quad \text{where} \quad F_k(x) = \frac{1}{n_k}\sum_{d\in\mathcal{D}_k} \mathcal{L}(d;x)$$

Where a dataset $\mathcal{D}$ is partitioned among m agents, indexed by k.

The FedAVG framework depends on two parts, the central coordinator algorithm (Federated Avergaing), and a local client algorithm called ClientUpdate. Let S be the set of all m clients, indexed by k. Let F_k be the kth agent's objective function, defined over its private local dataset $\mathcal{D}_k$, with $|\mathcal{D}_k| = n_k$. Let t be the communication round, γ be the diminishing stepsize, and b be the batch size, dictated by the server. In each communication round t, each selected client runs ClientUpdate for E local epochs.

Algorithm 2 Federated Averaging

1: **Server Executes:**
2: Server chooses a random point x_0
3: **for** each round $t = 0, 1, \dots$ **do**
4: $c_t \leftarrow$ (sample from $\sim U[1/m, 1]$)
5: $m_t \leftarrow \lfloor c_t \cdot m \rfloor$
6: $S_t \leftarrow$ (random set of m_t clients)
7: **for** each client $k \in S_t$ **in parallel do**
8: $x_{t+1}^k \leftarrow \text{ClientUpdate}(k, x_t)$
9: **end for**
10: $n_t \leftarrow \sum_{k \in S_t} n_k$
11: $x_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{n_t} x_{t+1}^k$
12: **end for**
13: **return** x_{t+1}

14: **ClientUpdate**(k, x)**:** *// Run on client k*
15: $B \leftarrow$ (split $\mathcal{D}_k$ into batches of size b)
16: **for** each local epoch $i = 1 \dots E_t$ **do**
17: $x \leftarrow x - \gamma_t \nabla \mathcal{L}(B; x)$
18: **end for**
19: **return** x to server

A few things of note in the algorithm:

- The algorithm relies on a centralized server that handles coordination and information distribution between agents. Each agent has a local dataset which is held privately and never uploaded to the server. A graph representation of this network would be a star network with $m + 1$ nodes.
- In each communication round, the server chooses a nonempty subset of clients S_t to run ClientUpdate, which is effectively a local SGD operation for the kth agent. If $c_t = 1$, then $S_t = S$. Also, each local dataset is sampled to create minibatches for the SGD update in line 17. If $b = \infty$, then $B = \mathcal{D}_k$ for all k. If both $c_t = 1$ and $b = \infty$, this corresponds to full-batch SGD being run on every agent [3]. This provides some robustness against individual agent failure, but the algorithm is still vulnerable to the loss of the server.
- FedAVG solves the consensus problem by taking a weighted average of all participating agent solutions and using it as its global solution vector. This is an easy and effective way of solving the consensus problem, and in the case of FedAVG's method, also allows for convergence to a global solution. It can be shown that FedAVG converges at a rate $O\left(\frac{1}{T}\right)$ where T is the total number of gradient computations across a particular agent, i.e. $T = \sum_t E_t$ [2]. That is, FedAVG achieves sublinear convergence in the best case.

While Federated Averaging enjoys many benefits and is commonly used in modern distributed deep learning applications, there are several major limitations to the framework, mostly due to the need of a coordinating master node. This makes the method vulnerable to failure of the master node, and often inflexible with respect to changes in graph connections [5]. In real-world situations, such as in mobile phone networks, these limitations cannot be avoided for the Federated Averaging

framework. One type of framework that attempts to address the weakness of Federated Averaging is called Gradient Tracking.

5 DSGT

The Gradient Tracking family is a collection of distributed optimization frameworks that attempt eliminate the need for a centralized coordinator. Instead, agents in a network communicate only with their immediate neighbors, and pass gradient information in the form of a *gradient tracker* to converge to a global solution. Unlike in FedAVG, there is no fixed star network structure where one agent acts as a master node. Therefore, there is no central point of failure for the network. However, this also means that it is difficult to simply solve the consensus problem, as there is no way of averaging all agent solution vectors. This report will investigate a particular member of the Gradient Tracking family called Distributed Stochastic Gradient Tracking (DSGT), and provide a basic proof sketch of its convergence.

5.1 Problem Formulation

Just like in FedAVG, DSGT is capable of solving the following distributed machine learning optimization problem in (4), but it is also capable of solving a standard distributed formulation over m such as the problem in (2):

$$\min_{x\in\mathbb{R}^n} \qquad f(x) = \frac{1}{m}\sum_{i=1}^{m} f_i(x)$$

The following assumptions also must be made for the DSGT algorithm to be applicable [6]:

Assumption 1. *A set of agents $[m] = \{1, 2, \dots, m\}$ is represented over an undirected, connected netowrk $G(V, E)$, where $V = [m]$. Two agents $i, j \in V$ are neighbors if and only if there exists an edge $i \leftrightarrow j$ in E. Then, it is possible to define a weight matrix $\mathbf{W} = [w_{ij}] \in \mathbb{R}^{m\times m}$ satisfying the following:*

- $\mathbf{W}$ *is nonnegative, i.e. $w_{ij} \geq 0$ for all $i, j \in \mathcal{N}$.*
- $\mathbf{W}$ *is doubly stochastic, i.e. $\mathbf{W1} = \mathbf{1}^T\mathbf{W} = \mathbf{1}$, where $\mathbf{1} = [1, 1, \dots, 1]^T \in \mathbb{R}^m$*
- *$w_{ii} > 0$ for some $i \in \mathcal{N}$. $w_{ij} > 0$ if and only if there exists an edge $i \leftrightarrow j$.*

Assumption 2. *For all $i \in [m]$, $f_i \in \mathbb{R}^n \to \mathbb{R}$ is μ-strongly convex with L-Lipschitz continuous gradients, private to agent i. This means that there is a unique solution $x^* \in \mathbb{R}^n$ to this problem.*

Assumption 3. *For all $i \in [m]$, agent i has access to a local stochastic oracle that allows it to obtain noisy gradient samples of the form $\nabla f_i(x, \xi_i)$, where $\xi_i \in \mathbb{R}^d$ is a d-dimensional random variable. $\nabla f_i(x, \xi_i)$ must satisfy the following:*

$$\mathbb{E}[\nabla f_i(x, \xi_i) \mid x] = \nabla f_i(x), \qquad \text{and} \qquad \mathbb{E}[\|\nabla f_i(x, \xi_i) - \nabla f_i(x)\|^2 \mid x] \leq \sigma^2 \text{ for some } \sigma > 0$$

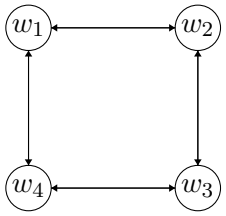
Assumption 4. *For all $i \in [m]$, random vectors ξ_i are independent from each other. Moreover, given a sequence of samples $\{\xi_{i,0}, \xi_{i,1}, \xi_{i,2}, \dots\}$, each $\xi_{i,k}$ is independently and identically distributed from sampling ξ_i.*

5.2 Building Weight Matrices from Graphs

DSGT relies on a doubly-stochastic weight mixing matrix $\mathbf{W}$ created from the agent network G to disseminate important gradient and soluton information among agents. Therefore, it is necessary to find efficient ways of generating large weight matrices satisfying the requirements in assumption 1. A very simple algorithm for constructing $\mathbf{W}$, assuming that every vertex in G has an equal number of neighbors, is:

$$w_{ij} = \begin{cases} \frac{1-\beta}{|N_G(i)|} & \text{if } j \in N_G(i) \\ \beta & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Where $0 \leq \beta \leq 1$ and $N_G(i)$ denotes the set of neighbors of node i.



$$\mathbf{W} = \begin{bmatrix} 0.6 & 0.2 & 0 & 0.2 \\ 0.2 & 0.6 & 0.2 & 0 \\ 0 & 0.2 & 0.6 & 0.2 \\ 0.2 & 0 & 0.2 & 0.6 \end{bmatrix}$$

Figure 1: an ring network and corresponding weight matrix generated using the rule in (5)

5.3 Algorithm Outline

Algorithm 3 Distributed Stochastic Gradient Tracking

1: **Input:** agents choose a doubly stochastic weight matrix $\mathbf{W}$ and set an initial step-size $\gamma_0 > 0$.
2: for all $i \in [m]$, agent i chooses a random initial point $x_{i,0} \in \mathbb{R}^n$
3: for all $i \in [m]$, agent i evaluates the initial gradient tracker $y_{i,0} \leftarrow \nabla f_i(x_{i,0}, \xi_{i,0})$
4: **for** each round $k = 0, 1, \ldots$ **do**
5: **for** each agent $i \in [m]$ **in parallel do**
6: $x_{i,k+1} \leftarrow \sum_{j=1}^{m} W_{ij} \left(x_{j,k} - \gamma_k y_{j,k}\right)$
7: $y_{i,k+1} \leftarrow \sum_{j=1}^{m} W_{ij} y_{j,k} + \nabla f_i(x_{i,k+1}, \xi_{i,k+1}) - \nabla f_i(x_{i,k}, \xi_{i,k})$
8: **end for**
9: **end for**

The update rules in lines 6 and 7 can be compactly represented by the following [6]:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{W}(\mathbf{x}_k - \gamma_k \mathbf{y}_k) \\ \mathbf{y}_{k+1} &= \mathbf{W}\mathbf{y}_k + \nabla F(\mathbf{x}_{k+1}, \xi_{k+1}) - \nabla F(\mathbf{x}_k, \xi_k) \end{aligned} \tag{6}$$

where $\mathbf{x} = [x_1, x_2, \ldots x_m]^T$, $\mathbf{y} = [y_1, y_2, \ldots y_m]^T$, and $\nabla F(\mathbf{x}, \xi) = [\nabla f_1(x_1, \xi_1), \ldots, \nabla f_m(x_m, \xi_m)]^T$.

The intent behind the DSGT algorithm is to use the weight matrix $\mathbf{W}$ to allow agents to spread gradient information across the network. Each agent i takes its own information and weights it against all if its immediate neighbors to produce a "neighbor-aware" local solution. Over time, it is expected that all agents in the network share enough information to converge globally to the optimal point x^*. As will be seen in the analysis, the gradient tracker $\mathbf{y}$ is very important in solving for both convergence and consensus.

The algorithm also highlights the difference between Federated Averaging and DSGT, and why DSGT is potentially a better optimization framework for large networks. The lack of a coordinator node means that the DSGT network is more robust against failure of any given agent. FedAVG does hedge against the possibility of agent failure by requiring only a subset of agents to do computation. However, FedAVG is still at risk of failure of the coordinator node. By eliminating this reliance entirely, DSGT is arguably better suited to massive distributed deployments.

5.4 Convergence and Consensus Analysis

Since there are so many parts to the DSGT algorithm, there are actually *three* separate convergence rates that must be analyzed. To do so, a few definitions/notations must be introduced:

1. $\bar{x} = \frac{1}{m}\mathbf{1}^T\mathbf{x}$ and $\bar{y} = \frac{1}{m}\mathbf{1}^T\mathbf{y} \in \mathbb{R}^{1\times n}$ are averages across the m rows of $\mathbf{x}$ and $\mathbf{y}$, respectively. $\frac{1}{m}\mathbf{1}^T$ is referred to as the averaging operator.
2. The filtration $\mathcal{F}_k = \bigcup_{i=1}^m \{x_{i,0}, \xi_{i,0}, \xi_{i,1}, \ldots, \xi_{i,k+1}\}$ for all $k \geq 1$ is essentially an efficient method of modeling all information available at time k, and is important in calculations of conditional expectation.
3. The filtration $\mathcal{F}_0 = \bigcup_{i=1}^m \{x_{i,0}\}$ is the initial starting state of the algorithm.

In particular, it can be shown that:

Theorem 1. *There exists constants E_1, E_2, $E_3 > 0$ such that for all $k \geq 1$:*

$$\mathbb{E}\left[\|\bar{x}_{k+1} - x^*\|^2\right] \leq \frac{E_1}{k+1} \tag{7}$$

$$\mathbb{E}\left[\|\mathbf{x}_{k+1} - \mathbf{1}\bar{x}_{k+1}\|^2\right] \leq \frac{E_2}{(k+1)^2} \tag{8}$$

$$\mathbb{E}\left[\|\mathbf{y}_{k+1} - \mathbf{1}\bar{y}_{k+1}\|^2\right] \leq E_3 \tag{9}$$

The above theorem provides some insight into the different rates of convergence of DSGT:

1. It is possible to achieve sublinear convergence of the averaged solution vector $\bar{x}_k = \frac{1}{m}\sum_{i=1}^m x_{i,k}$ to the optimal point x^*.
2. Sublinear convergence can be achieved for consensus.
3. It is not necessary for the individual gradient trackers $y_{i,k}$ to converge to $\bar{y}_k$. Simply having an initial bound on the distance between $\mathbf{y}_k$ and $\mathbf{1}\bar{y}_k$ is enough to allow for convergence of the solution vector.

Note that, in a centralized SGD algorithm (and centralized gradient descent), convergence holds at a rate $O\left(\frac{1}{k}\right)$. Points 1 and 2 together show that each agent converges to the same optimal value at the same asymptotic rate $O\left(\frac{1}{k}\right)$, and consensus occurs at an even faster $O\left(\frac{1}{k^2}\right)$. Therefore, DSGT is an efficient algorithm even for massively decentralized networks, and does not incur a penalty, up to a constant, when compared to centralized methods when the number of agents $m > 1$ [6].

Lemma 3. *Let assumptions 1-4 hold. Let $G(\mathbf{x}, \xi) = \frac{1}{m}\mathbf{1}^T\nabla F(\mathbf{x}, \xi)$, and $G(\mathbf{x}) = \frac{1}{m}\mathbf{1}^T\nabla F(\mathbf{x})$. Then the following hold for $k \geq 0$:*

1. *$\bar{y}_k = G(\mathbf{x}_k, \xi_k)$. In other words, the averaged gradient tracker $\bar{y}_k$ is the average of the stochastic gradient of all agents at iteration k.*

2. *$\mathbb{E}[\bar{y}_k \mid \mathcal{F}_k] = G(\mathbf{x}_k)$. This implies that $\bar{y}$ is an unbiased estimator of the average of the true gradients of all agents at iteration k.*

3. *$\mathbb{E}[\|\bar{y}_k - G(\mathbf{x}_k)\|^2 \mid \mathcal{F}_k] \leq \frac{\sigma^2}{m}$. This means that the variance of $\bar{y}_k$ is bounded by $\frac{\sigma^2}{m}$.*

4. *For any $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{m \times n}$, $\|G(\mathbf{u}) - G(\mathbf{v})\| \leq \frac{L}{\sqrt{m}}\|\mathbf{u} - \mathbf{v}\|$, where $\|\mathbf{A}\|$ denotes the Frobenius norm of matrix $\mathbf{A}$. This provides a bound on the error between any two sets of agent solution vectors during the runtime of the algorithm, by simply replacing $\mathbf{u}$ with $\mathbf{x}_k^1$ and $\mathbf{v}$ with $\mathbf{x}_k^2$.*

5. *$\|G(\mathbf{x}_k) - \nabla f(\bar{x}_k)\| \leq \frac{L}{\sqrt{m}}\|\mathbf{x}_k - \mathbf{1}\bar{x}_k\|$. This provides a bound on the error of average of the true gradients of all agents.*

6. *$\|\nabla f(\bar{x}_k)\| \leq L\|\bar{x}_k - x^*\|$. This provides a bound on the true gradient of the averaged solution vector at iteration k.*

Proof. Some proof ideas of lemma 3 are as follows:

1. The first proof proceeds by induction. For $k = 0$, using the initial state $\mathbf{y}_0$:

$$\bar{y}_0 = \frac{1}{m}\mathbf{1}^T\mathbf{y}_0 = \frac{1}{m}\sum_{i=1}^{m} y_{i,0} = \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x_{i,0}, \xi_{i,0}) = G(\mathbf{x}_0, \xi_0)$$

   Now, assume that $\bar{y}_k = G(\mathbf{x}_k, \xi_k)$ holds for some $k \geq 0$. According to the update rule in the algorithm:

$$\begin{aligned}
\frac{1}{m}\mathbf{1}^T\mathbf{y}_{k+1} &= \frac{1}{m}\mathbf{1}^T\mathbf{W}\mathbf{y}_k + \frac{1}{m}\mathbf{1}^T\nabla F(\mathbf{x}_{k+1}, \xi_{k+1}) - \frac{1}{m}\mathbf{1}^T\nabla F(\mathbf{x}_k, \xi_k) \\
\implies \bar{y}_{k+1} &= \frac{1}{m}\mathbf{1}^T\mathbf{y}_k + G(\mathbf{x}_{k+1}, \xi_{k+1}) - G(\mathbf{x}_k, \xi_k) \\
&= G(\mathbf{x}_k, \xi_k) + G(\mathbf{x}_{k+1}, \xi_{k+1}) - G(\mathbf{x}_k, \xi_k) \\
&= G(\mathbf{x}_k, \xi_k)
\end{aligned}$$

2. Using the result from part 1, and using the definition of conditional expectation as well as the result from Assumption 3:

$$\mathbb{E}[\bar{y}_k \mid \mathcal{F}_k] = \mathbb{E}[G(\mathbf{x}_k, \xi_k) \mid \mathcal{F}_k] = G(\mathbf{x}_k)$$

3. Using the definitions of $\bar{y}$ and the square of the L2-norm L2-norm:

$$\begin{aligned}
\mathbb{E}[\|\bar{y}_k - G(\mathbf{x}_k\|^2 \mid \mathcal{F}_k] &= \frac{1}{m^2}\sum_{i=1}^{m}\mathbb{E}\left[\|\nabla f_i(x_{i,k}, \xi_{i,k}) - \nabla f_i(x_{i,k})\|^2 \mid \mathcal{F}_k\right] \\
&\quad + \frac{1}{m^2}\sum_{i \neq j}\mathbb{E}\left[\left(\nabla f_i(x_{i,k}, \xi_{i,k}) - \nabla f_i(x_{i,k})\right)^T\left(\nabla f_j(x_{j,k}, \xi_{j,k}) - \nabla f_j(x_{j,k})\right) \mid \mathcal{F}_k\right] \\
&\leq \frac{1}{m^2}\sum_{i=1}^{m}\sigma^2 + 0 \\
&= \frac{\sigma^2}{m}
\end{aligned}$$

4. Squaring the left hand side and expanding, then using the Cauchy-Schwarz inequality:

$$\begin{aligned}\|G(\mathbf{u}) - G(\mathbf{v})\|^2 &= \left\| \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(u_i) - \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(v_i) \right\|^2 \\ &= \frac{1}{m^2}\left\| \sum_{i=1}^{m} \nabla f_i(u_i) - \nabla f_i(v_i) \right\|^2 \\ &\leq \frac{1}{m^2}\left(\sum_{i=1}^{m} \|\nabla f_i(u_i) - f_i(v_i)\|^2 \right)\left(\sum_{i=1}^{m} 1^2 \right) \\ &\leq \frac{1}{m}(L^2\|\mathbf{u}-\mathbf{v}\|^2)\end{aligned}$$

   Taking the square root of both sides, $\|G(\mathbf{u}) - G(\mathbf{v}\| \leq \frac{L}{\sqrt{m}}\|\mathbf{u}-\mathbf{v}\|$

5. The proof of 5 follows directly from the proof of 4, by setting $\mathbf{u} = \mathbf{x}_k$ and $\mathbf{v} = \mathbf{1}\bar{x}_k$.

6. Using the definition of $\nabla f(x) = \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x)$:

$$\begin{aligned}\|\nabla f(\bar{x}_k)\| &= \left\| \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(\bar{x}_k) \right\| \\ &\leq \frac{1}{m}\sum_{i=1}^{m} \|\nabla f_i(\bar{x}_k) - \nabla f_i(x^*)\| \\ &\leq \frac{1}{m}\sum_{i=1}^{m} L\|\bar{x}_k - x^*\| \\ &= L\|\bar{x}_k - x^*\|\end{aligned}$$

This concludes the proofs of Lemmas 3.1 through 3.6. □

Lemma 3 provides a start to the proof of Theorem 1, but a few additional Lemmas are required for a full proof of the theorem. These will not be proven, but some ideas as to their importance will be provided.

Lemma 4. *Let Assumption 2 hold, that is, let f_i be μ-strongly convex and L-smooth for all $i \in [m]$. Then, for any $\alpha \leq \frac{2}{\mu+L}$:*

$$\|\bar{x}_k - \alpha\nabla f(\bar{x}_k) - x^*\| \leq (1-\mu\alpha)\|\bar{x}_k - x^*\|$$

This lemma provides some intuition to choosing the optimal stepsize γ_k in the algorithm. The term $\bar{x}_k - \alpha\nabla f(\bar{x}_k)$ is very close to the update rule for the matrix of solution vectors $\mathbf{x}_{k+1} = \mathbf{W}(\mathbf{x}_k - \gamma_k\mathbf{y}_k)$. Using the result of Lemmas 3.2, 3.3, 3.5 and Lemma 4, the first recursion relation can be found:

Theorem 2. *Recursion relation for (7):*

$$\mathbb{E}[\|\bar{x}_{k+1} - x^*\|^2 \mid \mathcal{F}_k] \leq (1-\mu\gamma_k)\|\bar{x}_k - x^*\|^2 + \frac{\gamma_k L^2}{\mu m}(1+\mu\gamma_k)\|\mathbf{x}_k - \mathbf{1}\bar{x}_k\|^2 + \gamma_k^2\frac{\sigma^2}{m}$$

To create a second recursion relation, the following Lemma follows from the fact that $\mathbf{W}$ is doubly stochastic:

Lemma 5. *Let ρ_W denote the spectral norm of the matrix $\mathbf{W} - \frac{1}{m}\mathbf{1}\mathbf{1}^T$. Then $\rho_W < 1$ and*

$$\|\mathbf{W}\mathbf{u} - \mathbf{1}\bar{u}\| \leq \rho_W \|\mathbf{u} - \mathbf{1}\bar{u}\|$$

for all $\mathbf{u} \in \mathbb{R}^{m\times n}$, and $\bar{u} = \frac{1}{m}\mathbf{1}^T\mathbf{u}$.

Lemma 5 essentially describes a contraction operation on the difference between $\mathbf{W}\mathbf{u}$ and the averaged matrix $\mathbf{1}\bar{u}$. Being doubly stochastic, $\mathbf{W}$ is able to contract the differences between the rows of $\mathbf{u}$ to converge to a global average $\bar{u}$ across all rows. By definition, $\rho_W = \|\mathbf{W} - \frac{1}{m}\mathbf{1}\mathbf{1}^T\|_2 = \|\mathbf{A}\|_2$ is the maximum singular value of $\mathbf{A}$. Thus, the value of ρ_W depends on the connectivity of the graph G. If G is a very sparse graph (few connections between agents), then ρ_W approaches 1, and thus slower convergence is expected. If G is highly connected, then $\rho_W \ll 1$, and faster convergence is possible.

Using Lemmas 4 and 5, the second recursion relation can be found:

Theorem 3. *Recursion relation for (8):*

$$\mathbb{E}[\|\mathbf{x}_{k+1} - \mathbf{1}\bar{x}_{k+1}\|^2 \mid \mathcal{F}_k] \leq \frac{1+\rho_W^2}{2}\mathbb{E}[\|\mathbf{x}_k - \mathbf{1}\bar{x}_k\|^2] + \frac{\gamma_k^2(1+\rho_W^2)\rho_W^2}{1-\rho_W^2}\mathbb{E}[\|\mathbf{y}_k - \mathbf{1}\bar{y}_k\|^2]$$

Finally, a third recursion relation can also be found using the previous results:

Theorem 4. *Recursion relation for (9):*

$$\begin{aligned}\mathbb{E}[\|\mathbf{y}_{k+1} - \mathbf{1}\bar{y}_{k+1}\|^2 \mid \mathcal{F}_k] \leq{}& \left(1+\eta+(1+\eta^{-1})2L^2\gamma_k^2\right)\rho_W^2\mathbb{E}[\|\mathbf{y}_k - \mathbf{1}\bar{y}_k\|^2] \\ &+ (1+\eta^{-1})L^2(3L^2\gamma_k^2 + 2\|\mathbf{W}-\mathbf{I}\|^2)\mathbb{E}[\|\mathbf{x}_k - \mathbf{1}\bar{x}_k\|^2] \\ &+ (1+\eta^{-1})3mL^4\gamma_k^2\mathbb{E}[\|\bar{x}_k - x^*\|^2] \\ &+ \left((1+\eta^{-1})(3L^2\gamma_k^2 + 2m\gamma_k L + 2m) + 2\right)\sigma^2\end{aligned}$$

where $\eta > 0$ is an arbitrary scalar and $\mathbf{I}$ is the $m \times m$ identity matrix.

Note that the final recursion relation relies on an arbitrary scalar $\eta > 0$. To guarantee convergence of $\|\mathbf{y}_{k+1} - \mathbf{1}\bar{y}_{k+1}\|^2$, η must be chosen such that

$$\left(1+\eta+(1+\eta^{-1})2L^2\gamma_k^2\right)\rho_W^2 < 1$$

By finding a value of η that satisfies the inequality, an update rule for the stepsize γ_k can be found as well that guarantees convergence. Note that $\frac{1+\rho_W^2}{2} < 1$ can be to find a stricter upper bound for η

$$\left(1+\eta+(1+\eta^{-1})2L^2\gamma_k^2\right)\rho_W^2 \leq \frac{1+\rho_W^2}{2} \iff 1+\eta+(1+\eta^{-1})2L^2\gamma_k^2 \leq \frac{1+\rho_W^2}{2+\rho_W^2}$$

Choosing $\eta = \frac{1-\rho_W^2}{4\rho_W^2}$ gives

$$(1+\eta^{-1})2L^2\gamma_k^2 \leq \frac{1-\rho_W^2}{4\rho_W^2} \iff \gamma_k \leq \frac{1-\rho_W^2}{2L\rho_W\sqrt{2+6\rho_W^2}}$$

Therefore, a possible definition for γ_0 and γ_k are:

$$\gamma_0 = \frac{1-\rho_W^2}{2L\rho_W\sqrt{2+6\rho_W^2}} \qquad \gamma_k = \frac{\gamma_0}{k+1} \tag{10}$$

Finally, it is possible to show that the convergence results in Theorem 1 hold, using the recursive relations in Theorems 2, 3, and 4. A proof sketch is as follows:

Proof. Using induction on k, assume that that inequality (7) holds for k. Using the result from theorem 2:

$$\mathbb{E}[\|\bar{x}_{k+1} - x^*\|^2] \leq (1 - \mu\gamma_k)\frac{E_1}{k} + \frac{\gamma_k L^2}{\mu m}(1 + \mu\gamma_k)\frac{E_2}{k^2} + \gamma_k^2\frac{\sigma^2}{m}$$

Then, it suffices to show that

$$(1 - \mu\gamma_k)\frac{E_1}{k} + \frac{\gamma_k L^2}{\mu m}(1 + \mu\gamma_k)\frac{E_2}{k^2} + \gamma_k^2\frac{\sigma^2}{m} \leq \frac{E_1}{k+1}$$

Or, equivalently,

$$\frac{\gamma_k L^2}{\mu m}(1 + \mu\gamma_k)\frac{E_2}{k^2} + \gamma_k^2\frac{\sigma^2}{m} \leq \left(\frac{1}{k+1} - \frac{1 - \mu\gamma_k}{k}\right) E_1 \tag{11}$$

Similar inequalities can be produced for (8) and (9) as well:

$$\frac{\gamma_k^2(1 + \rho_W)^2\rho_W^2}{1 - \rho_W^2} E_3 \leq \left(\frac{1}{(k+1)^2} - \frac{1 + \rho_W^2}{2k^2}\right) E_2 \tag{12}$$

$$\begin{aligned} \Big((1 + \eta^{-1})(3L^2\gamma_k^2 + 2m\gamma_k L + 2m) + 2\Big)\sigma^2 \\ +(1 + \eta^{-1})3mL^4\gamma_k^2\frac{E_1}{k} \\ +(1 + \eta^{-1})L^2(3L^2\gamma_k^2 + 2\|\mathbf{W} - \mathbf{I}\|^2)\frac{E_2}{k^2} \leq \Big(1 - (1 + \eta + (1 + \eta^{-1})2L^2\gamma_k^2)\rho_W^2\Big) E_3 \end{aligned} \tag{13}$$

Solving the system of inequalities in (11), (12), and (13) yields values for E_1, E_2, and E_3. □

5.5 Numerical Results

Figure 2: sample images from the MNIST dataset

A common machine learning benchmark that can be solved by the DSGT framework is the MNIST classification problem. MNIST is a large database of handwritten digits (0-9), commonly used for image processing and computer vision tasks [1]. Each of the 70,000 images is made up of 28×28 gray pixels. The dataset is split into 60,000 training images and 10,000 testing images. For a classification problem, the logistic regression equation is used:

$$\begin{aligned} \min \quad & f(x) = \frac{1}{|\mathcal{D}_{\text{train}}|}\sum_{i \in \mathcal{D}_{\text{train}}} \ln(1 + \exp(-v_i u_i^T x)) \\ \text{subject to} \quad & x \in \mathbb{R}^n \end{aligned}$$

where the dataset $\mathcal{D}_{\text{train}} = \{(u, v) \in \mathbb{R}^{784} \times \{-1, +1\}\}$. In this case, each u is a size-784 vector of pixels, and v is a binary label (for example, is a given image a '5' or 'not 5'?). Minimizing logistic loss over the dataset corresponds to the machine learning model "learning" the optimal parameters x that best express the ability to discern images of '5' vs. images of 'not 5'.

For this numerical experiment, DSGT ($m = 4$ agents) is compared against a centralized SGD method ($m = 1$). DSGT is run with a batch size of 1, while SGD is run with a batch size of 4 to keep the number of noisy gradient evaluations the same across both trials. Each algorithm is run for 1,000 epochs. The global function value $f(\bar{x})$ and the consensus error $\|\mathbf{x} - \mathbf{1}\bar{x}\|$ every 100 epochs are plotted below:

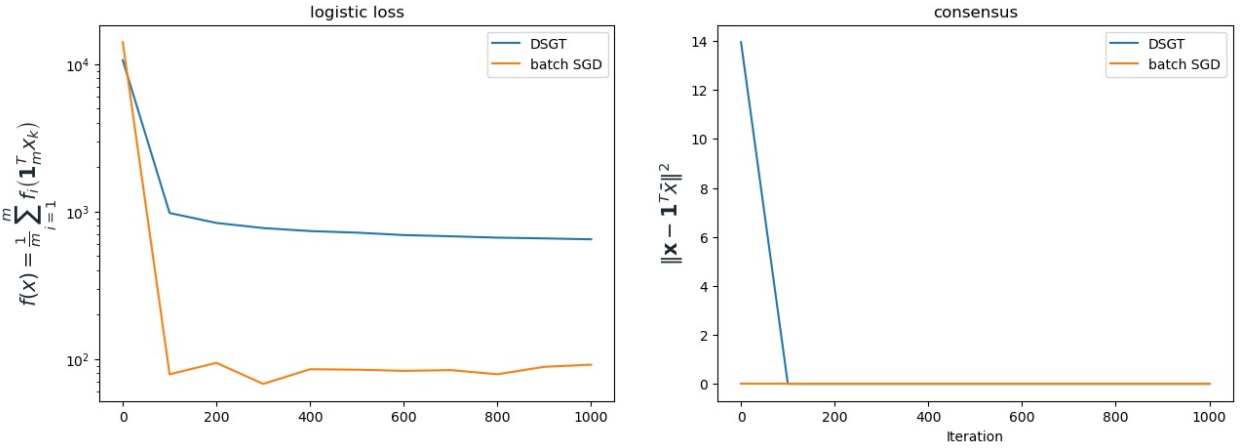


Figure 3: DSGT vs. SGD on MNIST, complete graph with 4 agents

The first thing to note from the above output is the consensus plot. Each agent in the network DSGT quickly converges to a consensus solution, as expected by the analysis in 5.4. Note that the plot on the right is not logarithmic, in order to show the batch SGD plot. Since batch SGD is a centralized method, its consensus plot is a constant 0.

Looking at the change in the loss function, it is clear that both DSGT and batch SGD converge at similar rates, and quickly stabilize at a solution. Most of the progress is made within the first 100 iterations. This is reflected in another samplepath looking at just the first 100 iterations. In this particular run, DSGT actually outperforms the centralized SGD method.

This numerical example shows that the DSGT framework is capable of providing similar performance as contemporary centralized methods, with little to no tradeoff in asymptotic performance, up to a constant. One place where DSGT suffers is runtime, where it is several orders of magnitude slower than SGD. SGD is able to complete 1000 iterations in under 5 seconds, while DSGT can take over 2-3 minutes.

The Github repository can be found at: `https://github.com/bzhanggg/ECE509_final`.

6 Concluding Remarks

DSGT is a privacy-preserving framework that allows for efficient training of machine learning models in massive decentralized distributed networks. Unlike Federated Averaging, DSGT does not rely on a central master node for coordination of agents, and thus is more resilient against network failures and attacks. DSGT also does not incur any penalty, up to a constant, in terms of convergence compared to centralized methods like SGD. DSGT's convergence to optimality is sublinear in $O\left(\frac{1}{k}\right)$,

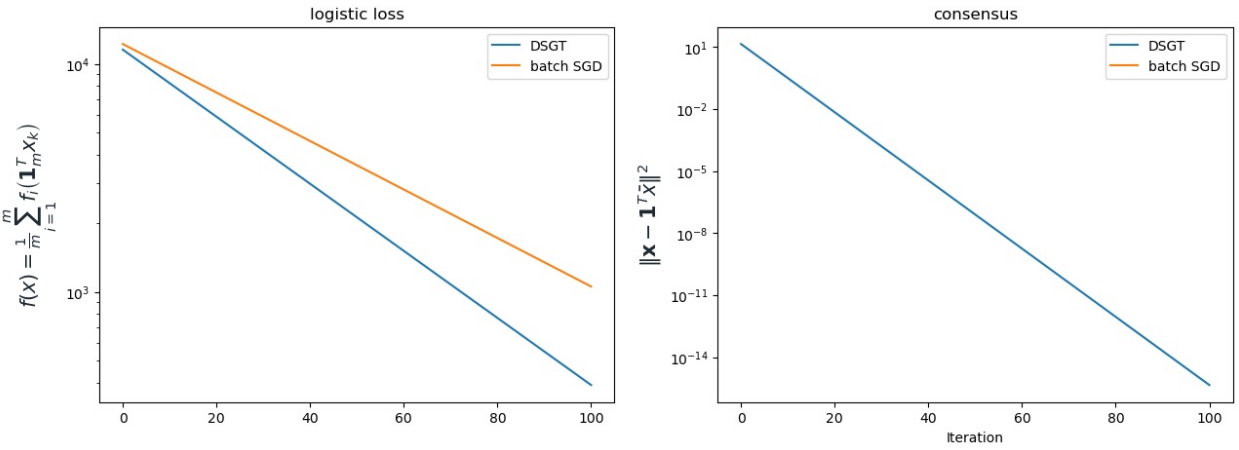


Figure 4: First 100 iterations of DSGT vs. SGD (the plot on the right is now logarithmic)

and convergence to consensus is also sublinear in $O\left(\frac{1}{k^2}\right)$.

Recent developments of DSGT have improved its flexibility to apply to different types of networks. For example, the Push-Pull method extends DSGT to directed networks, by splitting the weight matrix $\mathbf{W}$ into a row stochastic "pull" matrix $\mathbf{R}$ and a column stochastic "push" matrix $\mathbf{C}$ [4]. Additionally, DSGT can easily be adapted to time-varying networks by simply allowing for a changing weight matrix $\mathbf{W}_k$ with respect to iteration k [4]. Both of these developments have allowed for DSGT to be applied to more diverse environments and address different challenges.

Overall, the DSGT framework is a sound and flexible choice for distributed optimization problems where network robustness and data privacy are of the utmost importance.

References

[1] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[2] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, 2019.

[3] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[4] Angelia Nedić, Duong Thuy Anh Nguyen, and Duong Tung Nguyen. Ab/push-pull method for distributed optimization in time-varying directed networks. *Optimization Methods and Software*, pages 1–28, 2023.

[5] Angelia Nedić and Ji Liu. Distributed optimization for control. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(Volume 1, 2018):77–103, 2018.

[6] Shi Pu and Angelia Nedić. Distributed stochastic gradient tracking methods. *Mathematical Programming*, 187(1):409–457, 2021.

[7] Sebastian Ruder. An overview of gradient descent optimization algorithms. 2017.