Distributed Optimization and Federated Learning

Timothy Delille, Théophile Cabannes

April 2020

Abstract

This report presents a survey of consensus optimization and federated learning methods. Moreover, we introduce the Double Squeeze algorithm as a robust way to cut communications cost. We then compare global consensus optimization and general consensus optimization to a baseline gradient descent with backtracking line search applied to the multiclass logistic regression problem on the MNIST dataset.

1 Introduction

1.1 Motivations

The recent surge in demand for privacy-compliant applications might be seen as a challenge to current machine learning algorithms that rely on collecting large amount of data about users. Distributed learning seems to be a relevant way to perform machine learning algorithms while preserving the privacy of users. In distributed learning, machine learning algorithms are distributed on the devices that collect user information, so the user data are never communicated to a central server. Therefore, user information never exits user devices, leading to algorithms where privacy is ensured by design.

Distributing machine learning algorithms on several devices is sometime referred as **distributed learning**. Distributed optimization also enables the parallelization of optimization algorithms, which can improve the running time of the algorithms, as we show in this work.

1.2 Problem statement

We consider the following **regression problem**: given a set of n input data $(a_i)_{i\in[[1,n]]}$[1] and n corresponding output data $(b_i)_{i\in[[1,n]]}$, we estimate the output as a function h of the inputs that can be parametrized by x:

$$\hat{b}_i = h(x, a_i)$$

[1]Notation: $[[1, N]] = \mathbb{N} \cap [1, N]$, the integer numbers between 1 and N included.

Then we want to find the optimal parameter x such that the average differences between the output and the predicted output is minimized given a loss function L: $L(\hat{b}, b)$.

The problem can be cast as an optimization problem:

$$\begin{aligned} &\min_x \sum_{i=1}^{n} L(\hat{b}_i, b_i) \\ &\text{s.t. } \hat{b}_i = h(x, a_i), \quad \forall i \in [[1, N]] \end{aligned}$$

We assume that $x \to L(h(x, a_i), b_i)$ is convex for all i.

For this work, we consider the more general optimization problem:

$$\min f(x) = \sum_{i=1}^{N} f_i(x) \tag{1}$$

where f_i's are convex. The regression problem can be cast as (1) by splitting $[[1, n]]$ into N disjoint datasets $\mathcal{D}_1 \cup \mathcal{D}_2 \cdots \cup \mathcal{D}_N = [[1, n]]$, and defining f_i as:

$$f_i(x) = \sum_{k \in D_i} L(h(x, a_k), b_k)$$

Several paradigms of distributed learning exist (like Adaboost). In this work, we will focus on **federated learning** with a centralized server that distributes work to local workers.[2] Local workers perform local updates and communicate them back to the central server. We can write our federated learning approach as the following optimization problem:

$$\begin{aligned} &\min_{(x_i)_{i \in [[1,N]]}, z} \sum_{i=1}^{N} f_i(x_i) \\ &\text{subject to } x_i - z = 0, \quad \forall i \in [[1, N]] \end{aligned} \tag{2}$$

Assuming the f_i are computed by different workers, we refer to (2) as **consensus optimization** or **federated learning**.

1.3 Literature review

As a baseline, algorithms that solve (2) are often compared when the f_i computes the average value over their own dataset. This baseline is referred as the distributed average problem [JBA14].

[2] "In classical distributed learning setting when all workers participate in each round of communication [...]. In the federated setting only a small portion of workers participates in each round of communication." *Federated Learning with Communication Compression for Both Ups and Downs.* Anonymous Authors (under review for ICML).

Distributed learning and corresponding algorithms (including global form consensus optimization and general form consensus optimization) is explained in the textbook [BPC+11]. The usage of federated learning to compute the distributed average problem has been first introduced in [MMR+17]. Later, [TLZL19] introduces the concept of robustness to communication loss in federated learning with the `Double Squeeze` algorithm.

1.4 Results and outline

The report is presented as following: section 2 explains how to use alternative direction method of multipliers (ADMM) to solve consensus optimization problems. Section 3 introduces robustness of consensus optimization to communication loss with the double squeeze algorithm. Then section 4 compares four algorithms to perform multiclass logistic regression on the MNIST dataset:

1. Gradient descent with backtracking line search (consider as the baseline algorithm)
2. Global form consensus optimization
3. General form consensus optimization
4. Double squeeze algorithm (`FedAvg` [MMR+17] with compression)

The report shows that global and general form consensus optimization have comparable convergence rates with vanilla gradient descent with backtracking line search. In settings where gradient descent is not feasible on a single worker (e.g. large datasets or no access to the data), we can still achieve similar convergence rates using a distributed method. Global and general form consensus optimization both converge at the same rate, however it costs approximately 25% less per worker to perform computation.

All the code is available on this notebook `https://github.com/TimothyDelille/CS227C-Final-Project`

2 Consensus optimization and federated learning

2.1 Global form consensus optimization

The usage of the alternating direction method of multipliers (ADMM) to solve problem (2) is called global form consensus optimization [BPC+11].

Definition 2.1 (Augmented Lagrangian)**.** *The augmented Lagrangian of problem (2) is:*

$$\mathcal{L}_\rho(x, z, y) = \sum_{i=1}^{N}(f_i(x_i) + y_i^\top(x_i - z) + \frac{\rho}{2}\|x_i - z\|_2^2) = \sum_{i=1}^{N}\mathcal{L}_\rho^i(x_i, z, y)$$

with $\mathcal{L}_\rho^i(x_i, z, y) = f_i(x_i) + y_i^\top(x_i - z) + \frac{\rho}{2}\|x_i - z\|_2^2$.

Definition 2.2 (ADMM)**.** *The ADMM updates to solve the problem are:*

$$x_i^{k+1} = \arg\min_{x_i} \mathcal{L}_\rho(x, z^k, y^k) = \arg\min_{x_i} \mathcal{L}_\rho^i(x_i, z^k, y^k)$$
$$z_i^{k+1} = \arg\min_{z} \sum_{i=1}^{N} \mathcal{L}_\rho^i(x_i^{k+1}, z, y_i^k) = \frac{1}{N}\sum_{i=1}^{N}(x_i^{k+1} + \frac{1}{\rho}y_i^k)$$
$$y_i^{k+1} = y_i^k + \rho(x_i^{k+1} - z^{k+1})$$

Property 2.1. *As shown in [BPC+11], $\bar{y}$ is equal to zero after the first iteration and we have:*

$$z^{k+1} = \bar{x}^{k+1} = \frac{1}{N}\sum_{i=1}^{N} x_i^{k+1}$$

And we can substitute this in the previous ADMM updates.

Input: $\forall i \in [[1, N]]$: cost function f_i that includes user data, hyperparameter ρ, maximum number of iteration max_iter
$k = 0$, $x_k = 0$, $z_k = 0$, $y^k = 0$;
while *not converged* & $k \leq max\_iter$ **do**
  for $i \in [[1, N]]$ **do**
    $x_i^{k+1} = \arg\min_{x_i} f_i(x_i) + y_i^\top(x_i - z) + \frac{\rho}{2}\|x_i - z\|_2^2$;
  end
  $z^{k+1} = \frac{1}{N}\sum_{i=1}^{N} x_i^{k+1}$;
  $y_i^{k+1} = y_i^k + \rho(x_i^{k+1} - z^{k+1})$;
  $k = k + 1$;
end
return x^k

Algorithm 1: ADMM algorithm for consensus optimization

Remark 2.1 (Regularization)**.** *Adding a* ***regularization function*** $g(z)$ *to the problem only changes the z-update:*

$$\begin{aligned}
z^{k+1} &= \arg\min_z g(z) + \sum_{i=1}^{N}\Big(-(y_i^k)^\top z + (\rho/2)\|x_i^{k+1} - z\|_2^2\Big) \\
&= \arg\min_z g(z) - N(\bar{y}^k)^\top z + (\rho/2)\Bigg(\underbrace{\sum_{i=1}^{N}(x_i^{k+1})^\top(x_i^{k+1})}_{=N(\bar{x}^{k+1})^\top\bar{x}^{k+1} - \text{ terms independent of } z} - 2Nz^\top\bar{x}^{k+1} - z^\top z\Bigg) \\
&= \arg\min_z g(z) + (N\rho/2)\|z - \bar{x}^{k+1} - (1/\rho)\bar{y}^k\|_2^2 \\
&= prox_{\frac{1}{\rho N}, g}(\bar{x}^{k+1} + (1/\rho)\bar{y}^k)
\end{aligned}$$

For instance, in the case of L_1 regularization to the problem, the z-update becomes a soft-thresholding operation.

2.2 General Form Consensus Optimization

Suppose our dataset is very sparse, and some local dataset $\mathcal{D}_i$ contains features which are equal to zero for all entries in the dataset. It would be computationally more efficient to only train the model parameters which correspond to non-zero features for all entries in the dataset $\mathcal{D}_i$. The general form of consensus optimization appears when **each worker handles its own dataset**, as well as a **subset of model coefficients / features**. If in each dataset $\mathcal{D}_i$, every feature has a non-zero value, then this approach reduces to the global variable form we just saw.

Using the notation in [BPC$^+$11], the mapping $\mathcal{G}(i,j) = g$ means that the $j-th$ component of the local variable x_i corresponds to global variable component z_g, i.e. $(x_i)_j = z_g$. In the following figure from [BPC$^+$11], each edge represents a consensus constraint between a local component $(x_i)_j$ and a global variable $z_{\mathcal{G}(i,j)}$, which we denote $(\tilde{z}_i)_j$ (where $\tilde{z}_i$ contains the global components, limited to the ones x_i maps to).

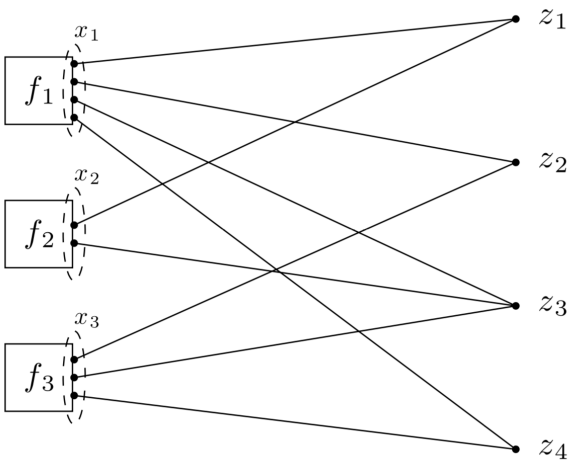

Figure 1: (from [BPC$^+$11]) $N = 3$ workers each have a local objective f_i whith a local variable of dimension n_i ($n_1 = 4, n_2 = 2, n_3 = 3$). Each edge represents a consensus constraint between a local variable component and a global variable component. In the figure above, $\mathcal{G}(2,2) = 3$ since $(x_2)_2$ maps to z_3

The vector $\tilde{z}_i$ represents the set of global variable components the local variable x_i maps to.

Definition 2.3 (Augmented Lagrangian)**.** *We can write augmented Lagrangian for this problem as:*

$$\mathcal{L}_\rho(x,z,y) = \sum_{i=1}^{N}(f_i(x_i) + y_i^\top(x_i - \tilde{z}_i) + \frac{\rho}{2}\|x_i - \tilde{z}_i\|_2^2)$$

with $x_i, y_i, \tilde{z}_i \in \mathbb{R}^{n_i}$. The ADMM updates are:

$$x_i^{k+1} = \arg\min_{x_i} f_i(x_i) + (y_i^k)^\top x_i + \frac{\rho}{2}\|x_i - \tilde{z}_i{}^k\|_2^2$$
$$z^{k+1} = \arg\min_{z} \sum_{i=1}^{N} -(y_i^k)^\top \tilde{z}_i + \frac{\rho}{2}\|x_i^{k+1} - \tilde{z}_i\|_2^2$$
$$y_i^{k+1} = y_i^k + \rho(x_i^{k+1} - \tilde{z}_i^{k+1})$$

Property 2.2. *The augmented Lagrangian is separable in its components and the z-update can be written as:*

$$z_g^{k+1} = \frac{\sum_{\mathcal{G}(i,j)=g} \left((x_i^{k+1})_j + (1/\rho)(y_i^k)_j\right.}{|(i,j) \text{ such that } \mathcal{G}(i,j) = g|}$$

Property 2.3. *Similarly than Property 2.1, we can show that after the first iteration: $\bar{y}_g = 0$.*
Therefore:

$$z_g^{k+1} = \frac{1}{k_g} \sum_{\mathcal{G}(i,j)=g} (x_i^{k+1})_j$$

Where: $k_g = |(i,j)$ such that $\mathcal{G}(i,j) = g|$, the number of local variable entries that are mapped to the global variable component z_g.

Remark 2.2. *Adding a regularization term to the objective reduces to a proximity operation on the z-update, just as in the global variable consensus case (remark 2.1).*

2.3 Federated Learning

Federated Learning is an emerging area of research in distributed optimization, in which multiple users (also referred to as *workers*) jointly train a machine learning model without having to reveal their data to other users. This is especially attractive in settings where companies want to benefit from each other's data without having to share their own or when exploiting the richness of mobile device's data to train models, for instance. [MMR+17] proposed the first `FedAvg` algorithm in which each worker conducts multiple local SGD iterations, and the central server (or *master*) pushes and pulls the model from a subset of workers at each iteration. The reason why we only use a subset of workers is to both decrease the dependency on unreliable workers (data poisoners or free riders) and cut communication cost (when communicating the model from workers to master (uplink) and master to worker (downlink)). Both these issues are active areas of research.

The problem can be formulated as:

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

Where n is the number of training examples.

The dataset is partitioned over N workers, each worker k having a dataset $\mathcal{D}_k$, therefore we can rewrite the objective as:

$$\min_x \sum_{k=1}^{N} \frac{|\mathcal{D}_k|}{n} F_k(x)$$

where $F_k(x) = \frac{1}{|\mathcal{D}_k|} \sum_{i \in \mathcal{D}_k} f_i(x)$

This can be viewed as the weighted sum of each worker's contribution (when the datasets are the same size: $|\mathcal{D}_k|/n = 1/N$). The varying size of data partitions can reflect the imbalanced contribution of individual users.

Moreover, under the assumption that the data is IID, we have:

$$\mathbb{E}_{\mathcal{D}_k} F_k(x) = f(x)$$

Where the expectation is taken over the dataset assigned to user k. However, a particular user's dataset might not be representative of the population ditribution.

The `FedAvg` algorithm is given below:

Data: $\forall i \in [[1, N]]$: cost function F_i, hyperparameter E
$x_1^0 = x_2^0 = \cdots = x_N^0 = z^0 = 0$;
for $t \in [[1, T]]$ **do**
  Randomly sample a subset $\mathcal{S}_t$ of users (or processors, or clients);
  Master (or root processor, or server) broadcasts all $z^\top$;
  for $i \in \mathcal{S}_t$ **do**
    $x_i^\top := z^\top$;
    $x_i^{t+1} :=$ E iterations of epoch-based Stochastic Gradient Descent on F_i;
    Broadcast x_i^{t+1} to master (or root processor, or server)
  end
  On master: $z^{t+1} := \sum_{k=1}^{N} \frac{|\mathcal{D}_k|}{n} x_i^{t+1}$;
end
return $z^\top$

Algorithm 2: Federated average algorithm

3 Robustness to communication loss: double squeeze algorithm

As communication can a major bottleneck in federated learning applications, a common strategy is to compress communication by using **quantization**[3]

[3]mapping from a large set of values to a smaller set (e.g. rounding or truncation). Performing quantization / compression introduces a quantization error.

of the model parameters that are communicated back and forth between the server and the clients.

`DoubleSqueeze` by [TLZL19] introduces **error compensation** to make up for information loss due to compression. In this setting, and under the IID assumption, the problem is formulated as follows:

$$\min_x f(x) = (1/N) \sum_{i=1}^{N} \mathbb{E}_{\zeta \sim \mathcal{D}_i} F(x; \zeta)$$

Each worker i computes a stochastic gradient $g^{(i)} = \nabla F(x, \zeta^{(i)})$ where $\zeta^{(i)}$ is drawn from $\mathcal{D}_i$ and the server aggregates the local gradients:

$$g = (1/N) \sum_{i=1}^{N} g^{(i)}$$

If we let Q_ω be a compression operator (which is not required to be unbiased), one should send the compressed, error-compensated vector:

$$Q_\omega[g^{(i)} + \delta^{(i)}]$$

where the compression error $\delta^{(i)}$ is updated as follows:

$$\delta^{(i)} = g^{(i)} + \delta^{(i)} - Q_\omega[g^{(i)} + \delta^{(i)}]$$

Compression options can be randomized quantization, k-bit (stochastic) quantization, clipping, top-k sparsification, etc...

Property 3.1 (Convergence of `DoubleSqueeze` algorithm [TLZL19])**.** *Under the following assumptions:*

- ***Lipschitz gradients**: $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \forall x, \forall y$*
- ***Bounded variance for stochastic gradient**:*
  $\mathbb{E}_{\zeta \sim \mathcal{D}_i} \|\nabla F(x; \zeta) - \nabla f(x)\|^2 \leq \sigma^2, \forall x, \forall i$
- *Q_ω **has bounded magnitude of error**:*

$$\begin{aligned} \mathbb{E}_\omega \|\delta_t^{(i)}\| &\leq \epsilon/2, \forall t, \forall i \qquad \textit{(worker's compression error at time } t\textit{)} \\ \mathbb{E}_\omega \|\delta_t\| &\leq \epsilon/2, \forall t \qquad \textit{(server's compression error at time } t\textit{)} \end{aligned}$$

`DoubleSqueeze` admits the same convergence rate as SGD, i.e.: $O\big(1/\sqrt{(T)}\big)$.

Remark 3.1. *In order to decrease dependency on unreliable workers, one could imagine picking a random subset of workers at each epoch, as in the original `FedAvg` algorithm.*

4 Comparison on multiclass logistic regression on MNIST dataset

In our application, we will interpret f_i in the problem (2) as the negative log-likelihood function for the model parameters given dataset $\mathcal{D}_i$. We can derive a probabilistic justification for the local x_i-updates in general form / global form ADMM for consensus optimization (see Appendix 6.2).

4.1 Problem Formulation

Let us apply the presented methods to a multiclass logistic regression problem on the MNIST dataset, which is commonly used as a benchmark in the statistical learning literature.

Suppose we want to classify an input $a \in \mathbb{R}^d$ into K classes. Multiclass logistic regression assigns each class k a score:

$$b_k = x_k^\top a$$

Where $x_k^\top$ is the $k-th$ row of the X matrix:

$$[b_1, \ldots, b_K] = Xa = \begin{pmatrix} x_1^\top \\ \vdots \\ x_K^\top \end{pmatrix} x$$

The estimated distribution $P(\hat{C}|a, X)$ is the softmax ditribution:

$$P(\hat{C} = j|a, X) = \frac{e^{b_j}}{\sum_{k=1}^K e^{b_k}}, j = 1, \ldots, K$$

Property 4.1 (Maximum likelihood estimation equivalent loss function)**.** *By minimizing the Kullback-Leibler Divergence over all training examples or, equivalently, performing Maximum Likelihood Estimation, we can derive the following loss function:*

$$f(X) = -\sum_{i=1}^{n}\sum_{j=1}^{K} \underbrace{\mathbb{P}(C_i = j)}_{\mathbb{I}_{c_i=j}} \ln \mathbb{P}(\hat{C}_i = j|a_i, X) = \sum_{i=1}^{N} f_i(X)$$

Proof can be found in Appendix 6.3.

Remark 4.1. *For each dataset $\mathcal{D}_i$:*

$$f_i(X) = -\sum_{k=1}^{n_i}\sum_{j=1}^{K} \mathbb{I}_{c_k=j} \ln \mathbb{P}(\hat{C}_k = j|a_k, X)$$

corresponds to the loss computed over dataset $\mathcal{D}_i = \{a_k | k \in \{n_{i-1}, \ldots, n_i\}\}$ ($n_0 = 1$ and $n_N = n$).

Property 4.2. *f is strictly convex, as a sum of strictly convex functions.*

4.2 Global Variable Consensus Optimization

As shown in the introduction, we can rewrite the problem as a consensus optimization problem:

$$\min \sum_{i=1}^{N} f_i(X_i) \text{ subject to } X_i = Z, i = 1, \ldots, N$$

The augmented Lagrangian is

$$\mathcal{L}_\rho(X_i^k, Z^k, Y_i^k) = f_i(X_i) + \text{Tr}((Y_i^k)^\top (X_i - Z^k)) + \frac{\rho}{2}\|X_i - Z^k\|_F^2$$

Using the fact that $Z^k = \bar{X}^k$, the ADMM updates are:

$$\begin{aligned}
X_i^{k+1} &= \arg\min_{X_i} \mathcal{L}_\rho(X_i, Z^k, Y_i^k) && \forall i \\
&= \arg\min_{X_i} f_i(X_i) + \text{Tr}((Y_i^k)^\top (X_i - Z^k)) + \frac{\rho}{2}\|X_i - Z^k\|_F^2 \\
Z^{k+1} &= \bar{X}^{k+1} \\
Y_i^{k+1} &= \arg\min_{Y_i} \mathcal{L}_\rho(X_i^{k+1}, Z^{k+1}, Y_i) && \forall i \\
&= Y_i^k + \rho(X_i^{k+1} - Z^{k+1})
\end{aligned}$$

The right hand term in the X_i minimization step is differentiable and by first optimality condition:

$$0 = \nabla_{X_i} f_i(X_i^*) + Y_i^k + \rho(X_i^* - \bar{X}^k)$$

Property 4.3 (Gradient)**.**

$$\nabla_{X_i^l} f_i(X_i) = -\sum_{k=1}^{n_i} \big(\delta_{l,c_k} - \mathbb{P}(\hat{C}_k = l | X_i, a_k)\big) a_k \ \forall l = 1, \ldots, K$$

As no closed form solution exists for the X_i-update, we can perform T iterations of backtracking line search or another gradient method with some $\delta > 0$ tolerance on the Frobenius norm of the gradient. [EY17] discusses the convergence of ADMM in approximate settings.

4.3 General Form Consensus Optimization

The general form of consensus optimization presented earlier is particularly appealing for high-dimensional data that is **sparse** (such as images). As our dataset is only consisting of 0 (black) and 255 (white) pixel values, training will greatly benefit from this approach. For each dataset $\mathcal{D}_i$ we only train the subset of weights:

$$X_i \in \mathbb{R}^{K, n_i}$$

Where n_i represents the number of distinct pixel positions that are non-zero for all the images in the dataset $\mathcal{D}_i$. We also need to discard non-zero values in the training images when computing the probability $P(\hat{y}_i = k|x_i, X_i)$ (as well as the loss function and gradient), to match dimensions. This does not induce any change in the code for these functions, as we make use vectorization (see attached Jupyter Notebook for reference).
For $N = 100$ workers, this translate in approximately 25.5% less features to train (≈ 585 pixels instead of ≈ 784).

4.4 Results

The following graphs sum up the different convergence rates for test accuracy, as well as the cost (per worker).

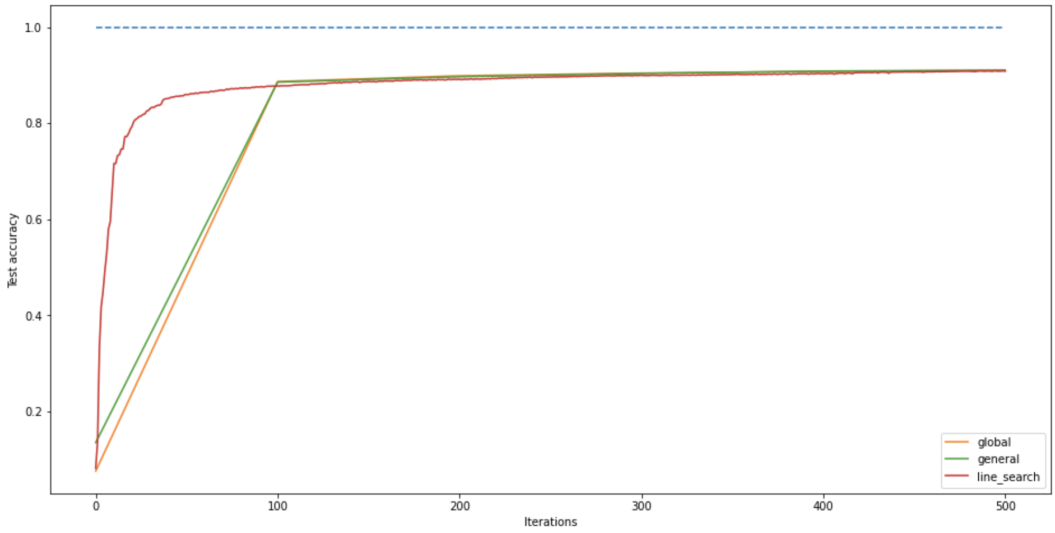

Figure 2: Each method converges to a $\approx 91\%$ test accuracy. Differences in convergence rates during the first iterations are merely due to the initialization of the model parameters

For each worker, we can see global and general form consensus optimization show the same convergence rate, as expected. The difference lies in the computation involved ($\approx 25\%$ less for each worker in our case). Moreover, their convergence rate is very close to that of vanilla gradient descent (i.e. only one worker). In more constraining settings such as large datasets or settings where we do not have access to the data (individual mobile devices e.g.), this means we can achieve a satisfying rate of convergence using a distributed method.

Unfortunately, we could not make the `DoubleSqueeze` algorithm converge, using this model. However, using a classic `LeNet` CNN architecture, `DoubleSqueeze`

converges to the same test set accuracy, while achieving a comparable convergence rate as Stochastic Gradient Descent.

The code is available at `https://github.com/TimothyDelille/CS227C-Final-Project`.

5 Conclusion and discussion

When training machine learning models, several questions arise, especially in the framework of distributed learning on mobile devices. For instance, *what if clients do not trust the server ?* or *what if the server is negligent, even though not malicious ?* [ASY+18] proposes a communication-efficient as well as differentially private distributed SGD algorithm, using a **Binomial mechanism** to incorporate noise in the local gradients. One reason binomial noise is an attractive alternative to the classical Gaussian mechanism is the fact that discrete random variables can be transmitted efficiently. The paper also includes an application to **Distributed Mean Optimization** which is closely related to the problem treated in federated learning.

As a conclusion, in this report we compared and implemented algorithms to perform consensus optimization and federated learning. We connect ADMM to the need of machine learning that preserves user privacy.

References

[ASY+18] Naman Agarwal, Ananda Theertha Suresh, Felix Xinnan X Yu, Sanjiv Kumar, and Brendan McMahan. cpsgd: Communication-efficient and differentially-private distributed sgd. In *Advances in Neural Information Processing Systems*, pages 7564–7575, 2018.

[BPC+11] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[EY17] Jonathan Eckstein and Wang Yao. Approximate admm algorithms derived from lagrangian splitting. *Computational Optimization and Applications*, 68(2):363–405, 2017.

[JBA14] Paulo Jesus, Carlos Baquero, and Paulo Sérgio Almeida. A survey of distributed data aggregation algorithms. *IEEE Communications Surveys & Tutorials*, 17(1):381–404, 2014.

[MMR+17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learn-

ing of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

[TLZL19] Hanlin Tang, Xiangru Lian, Tong Zhang, and Ji Liu. Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression. *arXiv preprint arXiv:1905.05957*, 2019.

6 Appendix

6.1 Background on duality, dual ascent, augmented lagrangian and ADMM

Consider the following convex optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x) \text{ subject to } Ax = b$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is convex and $A \in \mathbb{R}^{m \times n}$.
The Lagrangian of this problem is:

$$\mathcal{L}(x, y) = f(x) + y^\top (Ax - b)$$

and the dual function is:

$$\begin{aligned} g(y) &= \inf_x \mathcal{L}(x, y) \\ &= \{\inf_x f(x) + y^\top (Ax)\} - y^\top b \\ &= \sup_x -f(x) - y^\top (Ax) - y^\top b \\ &= f^*(-A^\top y) - b^\top y \end{aligned}$$

By definition of f^*, the convex conjugate of f.

The dual problem is therefore:

$$\max_{y \in \mathbb{R}^m} g(y)$$

If strong duality holds, the dual and primal problems have the same optimal values and if y^* is dual optimal, then any $x^* \in \arg\min_x \mathcal{L}(x, y^*)$ is primal optimal.

6.1.1 The dual ascent method

The Lagrangian is a concave function of y:

$$\begin{aligned} q(y) &= \min_x \mathcal{L}(x, y) \\ &= -\max_x -\mathcal{L}(x, y) \end{aligned}$$

By Danskin's theorem:

$$\begin{aligned} \partial(-q)(y) &= \Big\{ -\nabla_y \mathcal{L}(x, y) | x \in \arg\max_x \mathcal{L}(x, y) \Big\} \\ &= \Big\{ b - Ax | x \in \arg\max_x \mathcal{L}(x, y) \Big\} \end{aligned}$$

The updates of the dual ascent method are:

$$\begin{aligned} x^{k+1} &= \arg\min_x \mathcal{L}(x, y^k) \\ y^{k+1} &= y^k + \alpha^k \underbrace{d}_{\in \partial q(y^k)} \\ &= y^k + \alpha^k (Ax^{k+1} - b) \end{aligned}$$

With α^k the step size. One choice of step size, given by subgradient methods, could be $\alpha^k = \frac{\theta}{\sqrt{k}}$ for some $\theta \in \mathbb{R}$ (Recht & Wright Chap 9.2).

Now suppose that the objective f is separable, i.e.:

$$f(x) = \sum_{i=1}^{N} f_i(x_i)$$

where $x = (x_1, \ldots, x_N)$ partitioned as subvectors and $A = [A_1 \ldots A_N]$ so that $Ax = \sum_{i=1}^{N} A_i x_i$.
We can thus write the Lagrangian as:

$$\mathcal{L}(x, y) = \sum_{i=1}^{N} \mathcal{L}_i(x_i, y) = \sum_{i=1}^{N} (f_i(x_i) + y^\top A_i x_i - \frac{y^\top b}{N}$$

As this problem is also separable in x, it can be split in N separate problems, i.e.:

$$\begin{aligned} x_i^{k+1} &= \arg\min_{x_i} \mathcal{L}_i(x_i, y^k) \forall i = 1, \ldots, N \\ y^{k+1} &= y^k + \alpha^k (Ax^{k+1} - b) \end{aligned}$$

This is referred to as **dual decomposition**.

6.1.2 The Augmented Lagrangian

If we apply the proximal point method to our dual maximization problem, we get the following update:

$$y^{k+1} := \arg\max_y q(y) + \frac{1}{2\rho} \|y - y^k\|_2^2$$

This leads to the augmented Lagrangian (see Recht & Wright Chap. 10.5.2):

$$\mathcal{L}_\rho(x, y) = f(x) + y^\top (Ax - b) + \frac{\rho}{2} \|Ax - b\|_2^2$$

Applying dual descent to this problems yields the following updates:

$$\begin{aligned} x^{k+1} &= \arg\min_x \mathcal{L}_\rho(x, y^k) \\ y^{k+1} &= y^k + \rho (Ax^{k+1} - b) \end{aligned}$$

Although this method improves convergence properties of the dual ascent method, the augmented Lagrangian $\mathcal{L}_\rho$ is not separable even when f is separable because of the $L-2$ norm term that mixes the variables. Therefore, the x-minimization step cannot be performed in parallel.

6.1.3 Alternating Direction Method of Multipliers

The ADMM algorithm allows for decomposability while keeping greater convergence properties than the dual ascent method. The problem is formulated as:

$$\min f(x) + g(z) \text{ subject to } Ax + Bz = c$$

Where f and g are convex. ADMM consists of the following iterations:

$$\begin{aligned} x^{k+1} &= \arg\min_x \mathcal{L}_\rho(x, z^k, y^k) \\ z^{k+1} &= \arg\min_z \mathcal{L}_\rho(x^{k+1}, z, y^k) \\ y^{k+1} &= y^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \end{aligned}$$

Separating the minimization over x and z in two steps instead of performing joint minimization is both computationally tractable and allows for decomposition when f or g are separable.

6.2 x_i-update in ADMM and Maximum A Posteriori Estimation

In our application, we interpret f_i in the problem (2) as the negative log-likelihood function for the model parameters given dataset $\mathcal{D}_i$. We can derive a probabilistic justification for the local x_i-updates in general form / global form ADMM for consensus optimization. If we assume a Gaussian prior[4] $\mathcal{N}(\bar{x}^k - \frac{1}{\rho}y_i^k, \rho^{-1}I)$ on our maximum a posteriori (MAP) estimate of

[4] [BPC+11] suggest $\mathcal{N}(\bar{x}^k + \frac{1}{\rho}y_i^k, \rho I)$ is the correct prior but the MAP estimation does not correspond to the x-update we derived above, although my calculations could be wrong

x, denoted $\hat{x}_{MAP} \in \mathbb{R}^d$ (with d parameters in the model), we have:

$$
\begin{aligned}
\hat{x}_{MAP} &= \arg\max_x p(x|\mathcal{D}_i) \\
&= \arg\max_x \frac{p(\mathcal{D}_i|x)p(x)}{p(\mathcal{D}_i)} \\
&= \arg\max_x \log(p(\mathcal{D}_i|x)) + \log(p(x)) \\
&= \arg\min_x -\log\left(\prod_{k=n_{i-1}}^{n_i} p(X_k, Y_k|x)\right) - \log(p(x)) \\
&= \arg\min_x -\sum_{k=n_{i-1}}^{n_i} \log\big(p(Y_i|X_i, x)p(X_i|x)\big) - \log p(x) \\
&= \arg\min_x \underbrace{\sum_{k=n_{i-1}}^{n_i} -\log(p(Y_i|X_i, x))}_{f_i(x)} - \underbrace{\sum_{k=n_{i-1}}^{n_i} \log(p(X_i|x))}_{\text{independent of } x} - \log p(x) \\
&= \arg\min_x f_i(x) + \frac{1}{2}\big(x - \bar{x}^k + \frac{1}{\rho}y_i^k\big)^\top(\rho^{-1}I)^{-1}\big(x - \bar{x}^k + \frac{1}{\rho}y_i^k\big) \\
&\qquad\qquad + \frac{1}{2}\log|\det \rho^{-1}I| + \frac{d}{2}\log 2\pi \\
&= \arg\min_x f_i(x) + \frac{\rho}{2}\left(\|x - \bar{x}^k\|_2^2 + \frac{2}{\rho}(x - \bar{x}^k)^\top y_i^k + \frac{1}{\rho^2}\|y_i^k\|_2^2\right) \\
&= \arg\min_x f_i(x) + (x - \bar{x}^k)^\top y_i^k + \frac{\rho}{2}\|x - \bar{x}^k\|_2^2 \\
&= \arg\min_x \mathcal{L}_\rho^i(x, z^k, y^k) \\
&= x_i^{k+1}
\end{aligned}
$$

which corresponds to the x_i-update for each worker i.

6.3 Proof of Property 4.1

Proof. Denoting $\mathbb{I}_{i=j}$ as $\delta_{i,j}$:

$$\begin{aligned}
\hat{W} &= \arg\max_W \mathbb{P}(\hat{Y}_i = y_1, \dots, \hat{Y}_n = y_n | x_1, \dots, x_n, W) \\
&= \arg\max_W \prod_{i=1}^{n} \mathbb{P}(\hat{Y}_i = j | x_I, W) \\
&= \arg\max_W \prod_{i=1}^{n} \prod_{j=1}^{K} \mathbb{P}(\hat{Y}_i = j | x_I, W)^{\delta_{j,y_i}} \\
&= \arg\max_W \sum_{i=1}^{n} \sum_{j=1}^{K} \delta_{j,y_i} \ln \mathbb{P}(\hat{Y}_i = j | x_i, W) \\
&= \arg\max_W \sum_{i=1}^{n} \sum_{j=1}^{K} \delta_{j,y_i} \ln \left(\frac{e^{w_j^\top x_i}}{\sum_{k=1}^{K} e^{w_k^\top x_i}} \right) \\
&= \arg\min_W - \sum_{i=1}^{n} \sum_{j=1}^{K} \delta_{j,y_i} \ln \left(\frac{e^{w_j^\top x_i}}{\sum_{k=1}^{K} e^{w_k^\top x_i}} \right)
\end{aligned}$$

□

6.4 Proof of Property 4.3

Proof. Let us write the gradient for one row of W denoted w_l:

$$\begin{aligned}
\nabla_{w_l} L_i(W) &= \nabla_{w_l} - \sum_{i=1}^{n} \sum_{j=1}^{K} \delta_{j,y_i} \ln \left(\frac{e^{w_j^\top x_i}}{\sum_{k=1}^{K} e^{w_k^\top x_i}} \right) \\
&= - \sum_{i=1}^{n} \sum_{j=1}^{K} \delta_{j,y_i} \nabla_{w_l} \ln \left(\frac{e^{w_j^\top x_i}}{\sum_{k=1}^{K} e^{w_k^\top x_i}} \right) \\
&= - \sum_{i=1}^{n} \sum_{j=1}^{K} \delta_{j,y_i} \left(\nabla_{w_l} w_j^\top x_i - \nabla_{w_l} \ln \sum_{k=1}^{K} e^{w_k^\top x_i} \right) \\
&= - \sum_{i=1}^{n} \sum_{j=1}^{K} \delta_{j,y_i} \left(\delta_{l,j} - \frac{e^{w_j^\top x_i}}{\sum_{k=1}^{K} e^{w_k^\top x_i}} \right) x_i \\
&= - \sum_{i=1}^{n} (\delta_{l,y_i} - \mathbb{P}(\hat{y}_i = l)) x_i
\end{aligned}$$

Since $\sum_{j=1} \delta_{j,y_i} \delta_{l,j} = \delta_{l,y_i}$. □

6.5 ADMM updates: error introduced in approximate settings

In the context of our application to the MNIST dataset, let us denote:

$$L = \sup\|\nabla_{W_i} L_i(W_i)\| = \|n_i \vec{1}_{d\times K}\|_F = n_i\sqrt{Tr(K\vec{1}_{d\times d})} = n_i\sqrt{Kd}$$

Indeed, using the expression for $L_i(W_i)$, $\sup \nabla_{W_i} L_i(W_i)$ is attained for $\delta_{l,y_k} = 0$, $\mathbb{P}(\hat{y_k} = l) = 1$ and $x_k = 1$ (assuming the input data is rescaled to be between 0 and 1) $\forall k = 1, \ldots, n_i, \forall l = 1, \ldots, K$.

Therefore, L_i is L-Lipschitz with $L = n_i\sqrt{Kd}$ and we have guarantees on the averaged iterate $\bar{W_i}^\top = \frac{\sum_{t=1}^{\top} \alpha^\top W_i^\top}{\sum_{t=1}^{\top} \alpha^\top}$ and, if we choose a decaying step size $\alpha^\top = \frac{1}{\sqrt{t}}$:

$$L_i(\bar{W_i}^\top) - L_i^* \leq \frac{2L^2 \log T + \|W_i^1 - W_i^*\|_2^2}{\sqrt{T}}$$

We can bound $\|W_i^1 - W_i^*\|_2$ by some constant C.

Moreover, by L-Lipschitz assumption:

$$\begin{aligned}
L_i^* &\leq L_i(\bar{W_i}^\top) + < \nabla_{W_i} L_i(\bar{W_i}^\top), (W_i^* - \bar{W_i}^\top) > + \frac{L}{2}\|\bar{W_i}^\top - W_i^*\|_F^2 \\
\Leftrightarrow L_i(\bar{W_i}^\top) - L_i^* &\geq < \nabla_{W_i} L_i(\bar{W_i}^\top), (\bar{W_i}^\top - W_i^*) > - \frac{L}{2}\|\bar{W_i}^\top - W_i^*\|_F^2 \\
&\geq -\frac{L}{2}\|\bar{W_i}^\top - W_i^*\|_F^2 - \|\bar{W_i}^\top - W_i^*\|_F \underbrace{\|\nabla_{W_i} L_i(\bar{W_i}^\top)\|_F}_{\leq \delta} \\
&\geq -\frac{L}{2}\|\bar{W_i}^\top - W_i^*\|_F^2 - \delta\|\bar{W_i}^\top - W_i^*\|_F
\end{aligned}$$

Therefore:

$$\begin{aligned}
L_i(\bar{W_i}^\top) - L_i^* \leq \epsilon &\Rightarrow -\frac{L}{2}\|\bar{W_i}^\top - W_i^*\|_F^2 - \delta\|\bar{W_i}^\top - W_i^*\|_F \leq \epsilon \\
&\Leftrightarrow \frac{L}{2}\|\bar{W_i}^\top - W_i^*\|_F^2 + \delta\|\bar{W_i}^\top - W_i^*\|_F + \epsilon \geq 0 \\
&\Leftrightarrow \|\bar{W_i}^\top - W_i^*\|_F \in \left[\frac{-\delta - \sqrt{\delta^2 - 2L\epsilon}}{L}, \frac{-\delta + \sqrt{\delta^2 - 2L\epsilon}}{L}\right] \\
&\Rightarrow \|\bar{W_i}^\top - W_i^*\|_F \leq \epsilon'
\end{aligned}$$

Where $\epsilon' = \frac{-\delta+\sqrt{\delta^2-2L\epsilon}}{L}$. After T iterations of gradient descent:

$$\begin{aligned}
\|\nabla_{W_i}\mathcal{L}_\rho(W_i^\top, \bar{W}^k, \lambda_i^k)\|_F &= \|\nabla_{W_i}L_i(W_i^\top) + \lambda_i^k + \rho(W_i^\top - \bar{W}^k)\|_F \\
&\leq \delta \\
&\Leftrightarrow 0 = \nabla_{W_i}L_i(W_i^*) + \lambda_i^k + \rho(W_i^* - \bar{W}^k) + \Delta, \|\Delta\|_F \leq \delta \\
&\Leftrightarrow 0 = \nabla_{W_i}L_i(W_i^*) + \lambda_i^k + \rho(W_i^* - \bar{W}^{k+1}) + \rho(\bar{W}^{k+1} - \bar{W}^k) + \Delta \\
&\Leftrightarrow 0 = \nabla_{W_i}L_i(W_i^*) + \lambda_i^{k+1} + \rho(\bar{W}^{k+1} - \bar{W}^k) + \Delta \\
\Leftrightarrow -\rho(\bar{W}^{k+1} - \bar{W}^k) - \Delta &= \nabla_{W_i}L_i(W_i^*) + \lambda_i^{k+1}
\end{aligned}$$

In the original ADMM updates, we have:

$$W_i^{k+1} = W_i^* = \arg\min_{W_i} \mathcal{L}_\rho(W_i, \bar{W}^k, \lambda_i^k)$$

Therefore, the left hand side can be seen as a residual for the dual feasibility condition $0 = \nabla_{W_i}L_i(W_i^{k+1}) + \lambda_i^{k+1}$. *Boyd & al., 2011* refers to it as the **dual residual**:

$$s^{k+1} = -\rho(\bar{W}^{k+1} - \bar{W}^k) - \Delta$$

The error term Δ corresponds to the residual gradient of the augmented Lagrangian with respect to W_i.

What is the error induced in $\bar{W}^{k+1}$?

We saw that $\|\bar{W_i}^{k+1} - W_i^{k+1,*}\|_F \leq \epsilon'$. Therefore:

$$\begin{aligned}
\|\bar{W}^{k+1} - \bar{W}^{k+1,*}\|_F &= \left\| \frac{1}{N}\sum_{i=1}^{N}(W_i^{k+1} - W_i^{k+1,*}) \right\|_F \\
&\leq \frac{1}{N} N\ \epsilon' = \epsilon'
\end{aligned}$$

By the triangle inequality and the fact that:

$$\|W_i^{k+1} - W_i^{k+1,*}\|_F \leq \|\bar{W_i}^{k+1} - W_i^{k+1,*}\|_F$$

since the objective value decreases at each iteration.

What is the error induced in λ_i^{k+1} ?

$$\begin{aligned}
\|\lambda_i^{k+1} - \lambda_i^{k+1,*}\|_F &= \rho\|W_i^{k+1} - W_i^{k+1,*} + \bar{W}^{k+1,*} - \bar{W}^{k+1}\ \|_F \\
&\leq \rho\epsilon'^2
\end{aligned}$$

by Cauchy-Schwarz' inequality.