NIPS-2017 Federated Multi-Task Learning

冯开宇

北京理工大学

2021 年 9 月 3 日

目录

Summary

TL;DR

In this work, we show that multi-task learning is naturally suited to handle the statistical challenges of this setting (federated learning), and propose a novel systems-aware optimization method, MOCHA (which generalizes the distributed optimization method COCOA), that is robust to practical systems issues.

What is Federated Learning

the Motivating of Federated Learning

e.g. (1)

Problem: Google want to train model with users' mobile data.
Possible solution Centralized learning.
Challenge e.g. GDPR.

e.g. (2)

Problem: Several hospitals want to jointly train a better model use medical data.
Possible solution Centralized learning (through aggregated data).
Challenge Laws or policies may forbid giving patients' data to others.

What is Federated Learning

Federated Learning vs. Distributed Learning

Federated learning is one of the distributed learning.

Main differences are:

- Users have control over their device and data.
- Worker nodes are unstable.
- **Communication cost is higher than computation cost**
- Data stored on worker nodes are **not IID**.
- The amount of data is severely imbalanced.

Which lead to ...

What is Federated Learning

Consideration (Research Direction) in Federated Learning System Design

- *Trade Computation for Communication* [1]
- Privacy
- *Robustness* [1]

[1] mainly focused in this paper.

In multi-task learning, the goal is to learn models for multiple related tasks simultaneously.

While the MTL literature is extensive, most MTL modeling approaches can be broadly categorized into two groups based on how they capture relationships amongst tasks. The first assumes that a clustered, sparse, or low-rank structure between the tasks is known a priori. A second group instead assumes that the task relationships are not known beforehand and can be learned directly from the data.

In comparison to learning a single global model, these MTL approaches can directly capture relationships amongst non-IID and unbalanced data, which makes them particularly well-suited for the statistical challenges of federated learning.

MOCHA - Model

The Problem We are Facing

- **Statistical Challenges**: The aim in federated learning is to fit a model to data,$\{\mathbf{X}_1, \ldots, \mathbf{X}_m\}$, generated by m distributed nodes. Each node, $t \in [m]$, collects data in a *non-IID* manner across the network, with data on each node being generated by a distinct distribution $\mathbf{X}_t \sim P_t$. The number of data points on each node,n_t , may also vary significantly, and there may be an underlying structure present that captures the relationship amongst nodes and their associated distributions.
- **Systems Challenges**: There are typically a large number of nodes, m, in the network, and communication is often a significant bottleneck. Additionally, the storage, computational, and communication capacities of each node may differ due to variability in hardware (CPU, memory), network connection (3G, 4G, WiFi), and power (battery level). These systems challenges, compounded with unbalanced data and statistical heterogeneity, make issues such as stragglers and fault tolerance significantly more prevalent than in typical data center environments.

MOCHA - Model

General Multi-Task Learning Setup

Given data $\mathbf{X}_t \in \mathbb{R}^{d \times n_t}$ from m nodes ($t \in [m]$), multi-task learning fits separate weight vectors $\mathbf{w}_t \in \mathbb{R}^d$ to the data for each task (node) through arbitrary convex loss functions ℓ^t (e.g., the hinge loss for SVM models). Many MTL problems can be captured via the following general formulation:

$$\min_{\mathbf{W},\boldsymbol{\Omega}} \left\{ \sum_{t=1}^{m} \sum_{i=1}^{n_t} \ell_t \left(\mathbf{w}_t^T \mathbf{x}_t^i, y_t^i \right) + \mathcal{R}(\mathbf{W}, \boldsymbol{\Omega}) \right\}$$

As an example, several popular MTL approaches assume that tasks form clusters based on whether or not they are related. This can be expressed via the following bi-convex formulation:

$$\mathcal{R}(\mathbf{W}, \boldsymbol{\Omega}) = \lambda_1 \operatorname{tr}\left(\mathbf{W} \boldsymbol{\Omega} \mathbf{W}^T \right) + \lambda_2 \|\mathbf{W}\|_F^2$$

With constants $\lambda_1, \lambda_2 > 0$, and where the second term performs L_2 regularization on each local model.

MOCHA - Model

Algorithm

Algorithm 1: MOCHA: Federated Multi-Task Learning Framework

```
Input: Data $\mathbf{X}_t$ from $t = 1, ..., m$ tasks, stored on one of $m$ nodes, and initial matrix $\mathbf{\Omega}_0$
Starting point $\boldsymbol{\alpha}^{(0)} := 0 \in \mathbb{R}^n$, $\mathbf{v}^{(0)} := 0 \in \mathbb{R}^b$
for iterations$i = 0, 1, ...$ do
    Set subproblem parameter $\sigma'$ and number of federated iterations, $H_i$
    for iterations $h = 0, 1, ..., H_i$ do
        for tasks $t \in \{1, 2, ..., m\}$ in parallel over $m$ nodes do
            call local solver, returning $\theta_t^h$-approximate solution $\Delta\boldsymbol{\alpha}_t$ of the local subproblem
            update local variables $\boldsymbol{\alpha}_t \leftarrow \boldsymbol{\alpha}_t + \Delta\boldsymbol{\alpha}_t$
            return updates $\Delta\mathbf{v}_t := \mathbf{X}_t \Delta\boldsymbol{\alpha}_t$
        Reduce: $\mathbf{v}_t \leftarrow \mathbf{v}_t + \Delta\mathbf{v}_t$
    Update $\mathbf{\Omega}$ centrally based on $\mathbf{w}(\boldsymbol{\alpha})$ for latest $\boldsymbol{\alpha}$
Central node computes $\mathbf{w} = \mathbf{w}(\boldsymbol{\alpha})$ based on the latest $\boldsymbol{\alpha}$
return $\mathbf{W} := [\mathbf{w}_1, ..., \mathbf{w}_m]$
```

MOCHA - Model

About $\boldsymbol{\Omega}$

- **Observation 1:** In general, (1) is not jointly convex in $\mathbf{W}$ and $\boldsymbol{\Omega}$, and even in the cases where (1) is convex, solving for $\mathbf{W}$ and $\boldsymbol{\Omega}$ simultaneously can be difficult.
- **Observation 2:** When fixing $\boldsymbol{\Omega}$, updating $\mathbf{W}$ depends on both the data $\mathbf{X}$, which is distributed across the nodes, and the structure $\boldsymbol{\Omega}$, which is known centrally.
- **Observation 3:** When fixing $\mathbf{W}$, optimizing for $\boldsymbol{\Omega}$ only depends on $\mathbf{W}$ and not on the data $\mathbf{X}$.

MOCHA - Model

Dual Problem

Let $n := \Sigma_{t=1}^{m} n_t$ and $X := Diag(X_1, \cdots, X_m) \in \mathbb{R}^{md\times n}$. With $\mathbf{\Omega}$ fixed, the dual of problem(1), defined with respect to dual variables $\boldsymbol{\alpha} \in \mathbb{R}^n$, is given by:

$$\min_{\boldsymbol{\alpha}} \left\{ \mathcal{D}(\boldsymbol{\alpha}) := \sum_{t=1}^{m} \sum_{i=1}^{n_t} \ell_t^* \left(-\boldsymbol{\alpha}_t^i\right) + \mathcal{R}^*(\mathbf{X}\boldsymbol{\alpha}) \right\}$$

where ℓ_t^* and $\mathcal{R}^*$ are the conjugate dual functions of ℓ_t and $\mathcal{R}$, respectively, and $\boldsymbol{\alpha}_t^i$ is the dual variable for the data point (x_t^i, y_t^i).

Assumption (1)

Given $\mathbf{\Omega}$, we assume that there exists a symmetric positive definite matrix $\mathbf{M} \in \mathbb{R}^{md \times md}$, depending on $\mathbf{\Omega}$, for which the function $\mathcal{R}$ is strongly convex with respect to $\mathbf{M}^{-1}$. Note that this corresponds to assuming that $\mathcal{R}^*$ will be smooth with respect to matrix $\mathbf{M}$.

Remark (1)

We can reformulate the MTL regularizer in the form of $\bar{\mathcal{R}}(\mathbf{w}, \bar{\mathbf{\Omega}}) = \mathcal{R}(\mathbf{W}, \mathbf{\Omega})$, where $\mathbf{w} \in \mathbb{R}^{md}$ is a vector containing the columns of $\mathbf{W}$ and $\bar{\mathbf{\Omega}} := \mathbf{\Omega} \otimes \mathbf{I}_{d \times d} \in \mathbb{R}^{md \times md}$. For example, we can rewrite the regularizer in (2) as $\bar{\mathcal{R}}(\mathbf{w}, \bar{\mathbf{\Omega}}) = tr(\mathbf{w}^T(\lambda_1 \bar{\mathbf{\Omega}} + \lambda_2 \mathbf{I})\mathbf{w})$. Writing the regularizer in this form, it is clear that it is strongly convex with respect to matrix $\mathbf{M}^{-1} = \lambda_1 \bar{\mathbf{\Omega}} + \lambda_2 \mathbf{I}$.

CoCoA (Fenchel-Rockafellar duality):

$$\begin{cases} \min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \left\{ \mathcal{O}_A(\boldsymbol{\alpha}) := f(A\boldsymbol{\alpha}) + g(\boldsymbol{\alpha}) \right\}, & \text{(1a)} \\ \min_{\mathbf{w}\in\mathbb{R}^m} \left\{ \mathcal{O}_B(\mathbf{w}) := f^*(\mathbf{w}) + g^*(-A^\top \mathbf{w}) \right\}, & \text{(1b)} \end{cases}$$

MOCHA:

$$\begin{cases} \min_{\mathbf{W},\boldsymbol{\Omega}} \left\{ \sum_{t=1}^{m} \sum_{i=1}^{n_t} \ell_t \left(\mathbf{w}_t^T \mathbf{x}_t^i, y_t^i \right) + \mathcal{R}(\mathbf{W}, \boldsymbol{\Omega}) \right\}, & \text{(2a)} \\ \min_{\boldsymbol{\alpha}} \left\{ \mathcal{D}(\boldsymbol{\alpha}) := \sum_{t=1}^{m} \sum_{i=1}^{n_t} \ell_t^* \left(-\boldsymbol{\alpha}_t^i \right) + \mathcal{R}^*(\mathbf{X}\boldsymbol{\alpha}) \right\}, & \text{(2b)} \end{cases}$$

1a matches 2b, and 1b matches 2a. ℓ_t matches g and vice versa.

MOCHA

Data-local quadratic subproblems

The following data-local subproblems is a careful quadratic approximation of the dual problem to separate computation across the nodes. These subproblems find updates $\Delta\boldsymbol{\alpha}_t \in \mathbb{R}^{n_t}$ to the dual variables in $\boldsymbol{\alpha}$ corresponding to a single node t, and only require accessing data which is available locally, i.e., $\mathbf{X}_t$ for node t. The t-th subproblem is given by:

$$\min_{\Delta\alpha_t} \mathcal{G}_t^{\sigma'}\left(\Delta\boldsymbol{\alpha}_t; \mathbf{v}_t, \boldsymbol{\alpha}_t\right) := \sum_{i=1}^{n_t} \ell_t^*\left(-\boldsymbol{\alpha}_t^i - \Delta\boldsymbol{\alpha}_t^i\right) + \langle \mathbf{w}_t(\boldsymbol{\alpha}), \mathbf{X}_t\Delta\boldsymbol{\alpha}_t\rangle + \frac{\sigma'}{2}\left\|\mathbf{X}_t\Delta\boldsymbol{\alpha}_t\right\|_{\mathbf{M}_t}^2 + c(\boldsymbol{\alpha}),$$

where $c(\boldsymbol{\alpha}) := \frac{1}{m}\mathcal{R}^*(\mathbf{X}\boldsymbol{\alpha})$, and $\mathbf{M}_t \in \mathbb{R}^{d\times d}$ is the t-th diagonal block of the symmetric positive definite matrix $\mathbf{M}$.

Given dual variables $\boldsymbol{\alpha}$, corresponding primal variables can be found via $\mathbf{w}(\boldsymbol{\alpha}) = \nabla\mathcal{R}^*(\mathbf{X}\boldsymbol{\alpha})$, where $\mathbf{w}_t(\boldsymbol{\alpha})$ is the t-th block in the vector $\mathbf{w}(\boldsymbol{\alpha})$.

$$\min_{\Delta\alpha_t} \mathcal{G}_t^{\sigma'}(\Delta\boldsymbol{\alpha}_t; \mathbf{v}_t, \boldsymbol{\alpha}_t) := \sum_{i=1}^{n_t} \ell_t^*\left(-\boldsymbol{\alpha}_t^i - \Delta\boldsymbol{\alpha}_t^i\right)$$
$$+ \langle \mathbf{w}_t(\boldsymbol{\alpha}), \mathbf{X}_t\Delta\boldsymbol{\alpha}_t \rangle + \frac{\sigma'}{2}\|\mathbf{X}_t\Delta\boldsymbol{\alpha}_t\|_{\mathbf{M}_t}^2 + c(\boldsymbol{\alpha}),$$

Note that computing $\mathbf{w}(\boldsymbol{\alpha})$ requires the vector $\mathbf{v} = \mathbf{X}\boldsymbol{\alpha}$. The t-th block of $\mathbf{v}$, $\mathbf{v}_t \in \mathbb{R}^d$, is the only information that must be communicated between nodes at each iteration.

One interesting aspect of MOCHA is that the method can be easily modified to accommodate the sharing of tasks among the nodes without any change to the local solvers. This property helps the central node to reduce the size of $\boldsymbol{\Omega}$ and the complexity of its update with minimal changes to the whole system.

Convergence Analysis

Approximately solvable subproblem

We define θ_t^h as a function of these factors, and assume that each node has a controller that may derive θ_t^h from the current clock cycle and statistical/systems setting. θ_t^h ranges from zero to one, where $\theta_t^h = 0$ indicates an exact solution to $\mathcal{G}_t^{\sigma'}(\cdot)$and $\theta_t^h = 1$ indicates that node t made no progress during iteration h (which we refer to as a *dropped node*).

$$\theta_t^h := \frac{\mathcal{G}_t^{\sigma'}\left(\Delta\boldsymbol{\alpha}_t^{(h)}; \mathbf{v}^{(h)}, \boldsymbol{\alpha}_t^{(h)}\right) - \mathcal{G}_t^{\sigma'}\left(\Delta\boldsymbol{\alpha}_t^{\star}; \mathbf{v}^{(h)}, \boldsymbol{\alpha}_t^{(h)}\right)}{\mathcal{G}_t^{\sigma'}\left(\mathbf{0}; \mathbf{v}^{(h)}, \boldsymbol{\alpha}_t^{(h)}\right) - \mathcal{G}_t^{\sigma'}\left(\Delta\boldsymbol{\alpha}_t^{\star}; \mathbf{v}^{(h)}, \boldsymbol{\alpha}_t^{(h)}\right)}$$

where $\Delta\boldsymbol{\alpha}_t^*$ is the minimizer of subproblem $\mathcal{G}_t^{\sigma}(\cdot; \mathbf{v}^{(h)}, \boldsymbol{\alpha}^t)$.

Assumption (2)

Let $\mathcal{H}_h := (\boldsymbol{\alpha}^{(h)}, \boldsymbol{\alpha}^{(h-1)}, \cdots, \boldsymbol{\alpha}^{(1)})$ *be the dual vector history until the beginning of iteration* h*, and define* $\Theta_t^h := \mathbb{E}[\theta_t^h | \mathcal{H}_h]$*. For all tasks* t *and all iterations* h*, we assume* $p_t^h := \mathcal{P}[\theta_t^h = 1] \leq p_{max} < 1$ *and* $\hat{\Theta}_t^h := \mathbb{E}[\theta_t^h | \mathcal{H}_h, \theta_t^h < 1] \leq \theta_{max} < 1$.

This assumption states that at each iteration, the probability of a node sending a result is non-zero, and that the quality of the returned result is, on average, better than the previous iterate.

MOCHA - Convergence Analysis

Convergence

Theorem (1)

Assume that the losses ℓ_t are $(1/\mu) - smooth$. Then, under Assumptions 1 and 2, there exists a constant $s \in (0, 1]$ such that for any given convergence target $\epsilon_{\mathcal{D}}$, choosing H such that

$$H \geq \frac{1}{(1-\bar{\Theta})s} \log \frac{n}{\epsilon_{\mathcal{D}}}$$

will satisfy $\mathbb{E}\left[\mathcal{D}\left(\boldsymbol{\alpha}^{(H)}\right) - \mathcal{D}\left(\boldsymbol{\alpha}^{\star}\right)\right] \leq \epsilon_{\mathcal{D}}$.

Here, $\bar{\Theta} := p_{max} + (1-p_{max})\Theta_{max} < 1$. While Theorem 1 is concerned with finite horizon convergence, it is possible to get asymptotic convergence results, i.e., $H \to \infty$, with milder assumptions on the stragglers; see Corollary 8 in the Appendix for details.

Convergence Analysis

When the loss functions are non-smooth, e.g., the hinge loss for SVM models, we provide the following sub-linear convergence for L-Lipschitz losses.

Theorem (2)

If the loss functions ℓ_t are L-Lipschitz, then there exists a constant σ, defined in (24), such that for any given $\epsilon_{\mathcal{D}} > 0$, if we choose

$$H \geq H_0 + \left[\frac{2}{(1-\bar{\Theta})} \max\left(1, \frac{2L^2\sigma\sigma'}{n^2\epsilon_{\mathcal{D}}}\right)\right]$$

$$\textit{with } H_0 \geq \left[h_0 + \frac{16L^2\sigma\sigma'}{(1-\bar{\Theta})n^2\epsilon_{\mathcal{D}}}\right], h_0 = \left[1 + \frac{1}{(1-\bar{\Theta})}\log\left(\frac{2n^2\left(D(\boldsymbol{\alpha}^\star)-D\left(\boldsymbol{\alpha}^0\right)\right)}{4L^2\sigma\sigma'}\right)\right]_+$$

then $\overline{\boldsymbol{\alpha}} := \frac{1}{H-H0}\sum_{h=H_0+1}^{H} \boldsymbol{\alpha}^{(h)}$ *will satisfy* $\mathbb{E}\left[\mathcal{D}(\overline{\boldsymbol{\alpha}}) - \mathcal{D}\left(\boldsymbol{\alpha}^\star\right)\right] \leq \epsilon_{\mathcal{D}}$.

Simulation

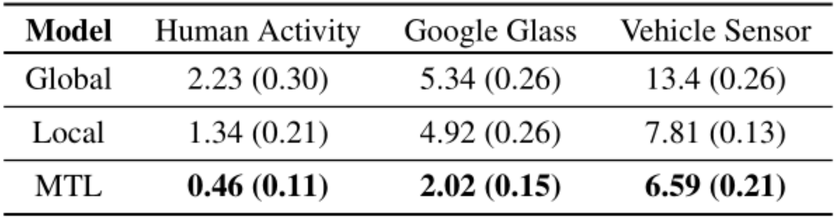
Datasets

- **Google Glass (GLEAM):** This dataset consists of two hours of high resolution sensor datacollected from 38 participants wearing Google Glass for the purpose of activity recognition.Following [41], we featurize the raw accelerometer, gyroscope, and magnetometer data into **180 statistical, spectral, and temporal features**. We **model each participant as a separate task, and predict between eating and other activities** (e.g., walking, talking, drinking).
- **Human Activity Recognition:** Mobile phone accelerometer and gyroscope data collected from **30 individuals**, **performing one of six activities**: walking, walking-upstairs, walking-downstairs,sitting, standing, lying-down. We use the provided 561-length feature vectors of time andfrequency domain variables generated for each instance. We **model each individual as a separate task and predict between sitting and the other activities**.

- **Vehicle Sensor:** Acoustic, seismic, and infrared sensor data collected from a distributed network of 23 sensors, deployed with the aim of classifying vehicles driving by a segment of road. Each instance is described by **50 acoustic and 50 seismic features**. We **model each sensor as aseparate task and predict between AAV-type and DW-type vehicles**.

Simulation

Multi-Task v.s. Global



Model	Human Activity	Google Glass	Vehicle Sensor
Global	2.23 (0.30)	5.34 (0.26)	13.4 (0.26)
Local	1.34 (0.21)	4.92 (0.26)	7.81 (0.13)
MTL	**0.46 (0.11)**	**2.02 (0.15)**	**6.59 (0.21)**

Figure: Average prediction error: Means and standard errors over 10 random shuffles.

Simulation

Statistical Heterogeneity a.k.a. Straggler Avoidance

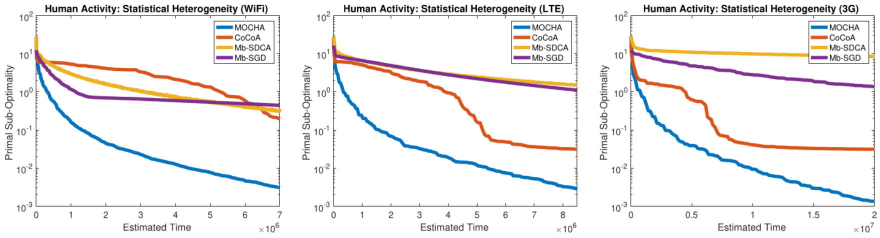


Figure: The performance of MOCHA compared to other distributed methods for the **W** update of equation in MTL settings. While increasing communication tends to decrease the performance of the mini-batch methods, MOCHA performs well in high communication settings. In all settings, MOCHA with varied approximation values, $^{h}_{t}$, performs better than without (i.e., naively generalizing COCOA), as it avoids stragglers from statistical heterogeneity.

Simulation

Systems Heterogeneity a.k.a. Data Variability

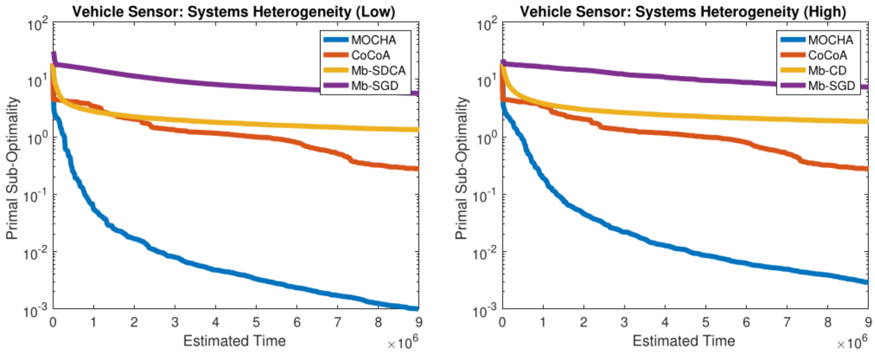


Figure: MOCHA can handle variability from systems heterogeneity.

Simulation

Tolerance to Dropped Nodes

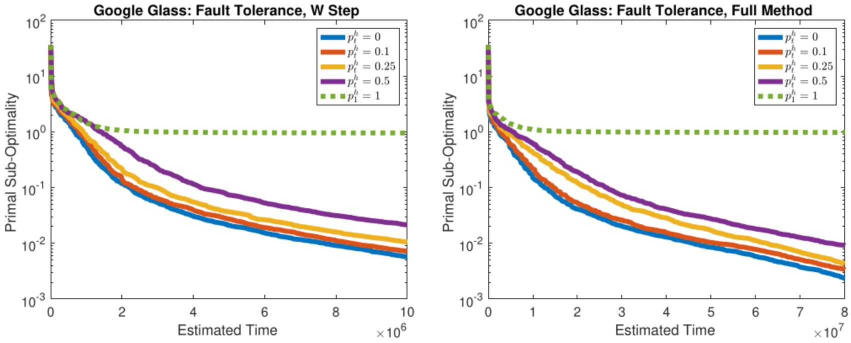


Figure: The performance of MOCHA is robust to nodes periodically dropping out (fault tolerance).

Discussion

in the Paper

While MOCHA does not apply to non-convex deep learning models in its current form, we note that there may be natural connections between this approach and "convexified" deep learning models in the context of kernelized federated multi-task learning.

Discussion

Revierwer

- The only major suggestion I would give is to possibly include discussion (or future work) addressing the limitation of requiring convex loss in real applications, how techniques presented in this work may benefit distributed nonconvex problems, and/or how they may benefit from ”convexified” stronger models (e.g. [1-4]).
- Not very novel (compared to COCOA) and just a mild variation over COCOA including the analysis.
- After the author feedback, it became clear that the synchronous updates and the changes to CoCoa have merit. Most of the intellectual contribution is on the convergence analysis rather than the learning algorithm, which is both a strong and a weak point.

  [1] Convex Deep Learning via Normalized Kernels, NIPS 2014

  [2] Convolutional Kernel Networks, NIPS 2014

  [3] Tensor Switching Networks, NIPS 2016

  [4] Convexified Convolutional Neural Networks, ICML 2017

Thanks!