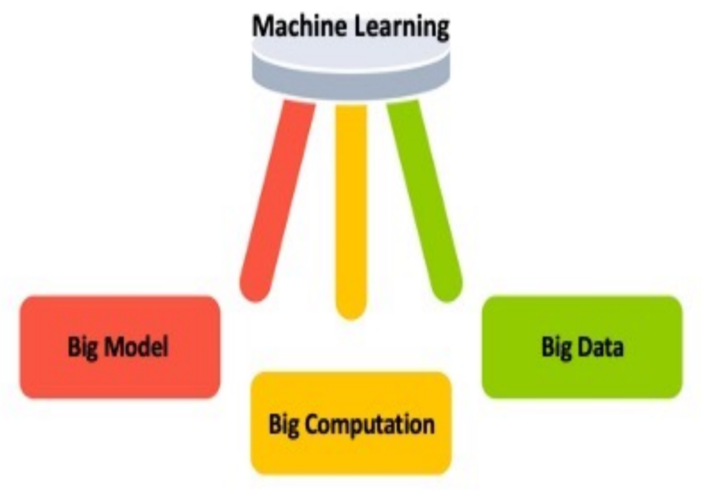
Finding Local Minima Efficiently in Decentralized Optimization

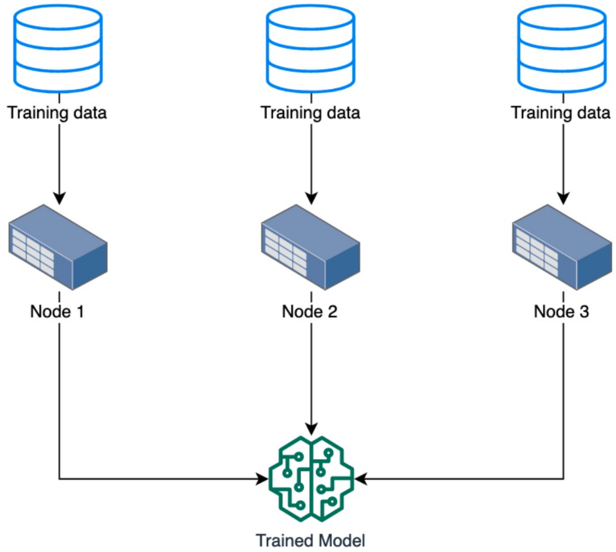
Wenhan Xian (wxian1@umd.edu)
Heng Huang* (heng@umd.edu)

UNIVERSITY OF MARYLAND

Introduction



Demand for big computation power

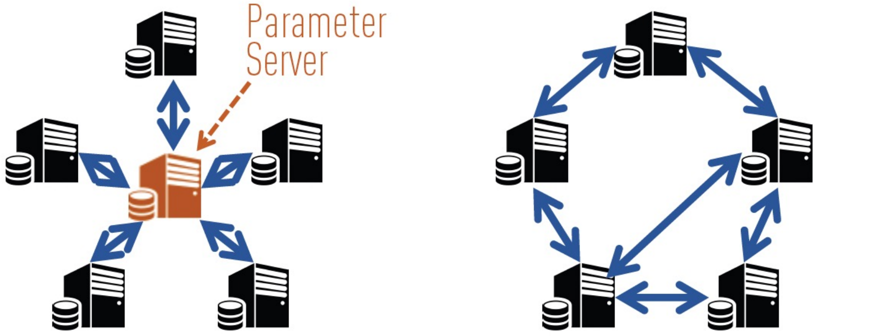


Distributed learning

UNIVERSITY OF MARYLAND

Decentralized Distributed Learning

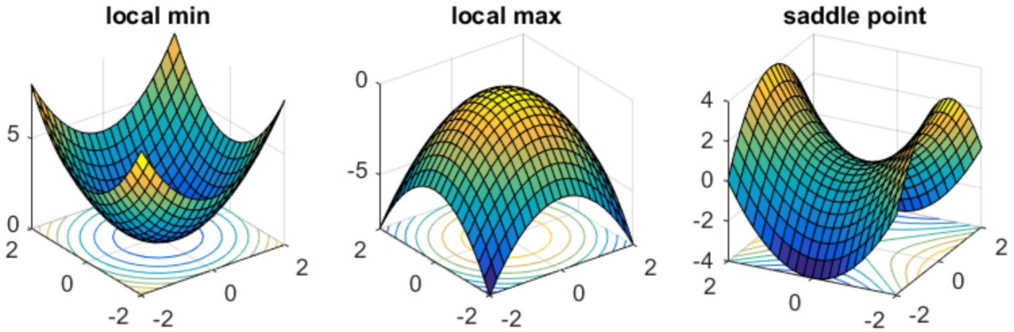
Decentralized (serverless) distributed learning is a class of distributed learning that trains models in parallel across multiple workers over a decentralized communication network. Each worker node only communicates with its neighbors.



(a) Centralized Topology (b) Decentralized Topology

Second-Order Optimality

- Escaping saddle point and finding local minima are core problems in conventional nonconvex optimization.
- Saddle point is a kind of first-order stationary point that can be reached by many gradient-based optimizers while it is not expected.



PDGT

Algorithm 1: PDGT algorithm

1: **Input:** $\mathbf{x}^0, \nabla f(\mathbf{x}^0), \epsilon, \gamma, \rho, \delta_1, \delta_2$

2: Set $\mathbf{x}_i = \mathbf{x}^0, \quad \mathbf{y}_i = \nabla f(\mathbf{x}^0), \quad T_1 = \tilde{\Theta}\left(\frac{f(\mathbf{x}^0) - f^*}{(1-\sigma)^2 \min\{\epsilon^2, \rho^2\}}\right), \quad T_2 = \tilde{\Theta}\left(\frac{d \log(1/\gamma\delta_2)}{\gamma^3}\right),$
$\eta_1 = \tilde{\Theta}\left((1-\sigma)^2\right), \quad \eta_2 = \tilde{\Theta}\left(\frac{\gamma^2}{d(1-\sigma)}\right), \quad \mathcal{R} = \tilde{\Theta}\left(\gamma^{\frac{3}{2}}\right), \quad B = \tilde{\Theta}\left(\gamma^3\right);$

3: Call $(\tilde{\mathbf{x}})$ = PDGT Phase I $(\underline{\mathbf{x}}, \underline{\mathbf{y}}, \eta_1, T_1, \delta_1)$;

4: Call $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, S)$ = PDGT Phase II $(\tilde{\mathbf{x}}, \eta_2, T_2, \mathcal{R}, B)$;

5: **if** $S = 1$ **then**

6: Return $\hat{\mathbf{x}}$ as a second-order stationary point and stop;

7: **else**

8: Set $\underline{\mathbf{x}} = \hat{\mathbf{x}}$, $\underline{\mathbf{y}} = \hat{\mathbf{y}}$ and go to Step 3;

9: **end if**

Algorithm 2: PDGT algorithm: Phase I

- **Input:** $\underline{\mathbf{x}}, \underline{\mathbf{y}}, \eta_1, T_1, \delta_1$
- **Initialization:** $\mathbf{x}^0 = \underline{\mathbf{x}}, \quad \mathbf{y}^0 = \underline{\mathbf{y}}$;
- **for** $r = 1, \ldots, T_1$ **do**
  - Compute $\mathbf{x}_i^r = \sum_{j \in \mathcal{N}_i} w_{ij} \mathbf{x}_j^{r-1} - \eta_1 \mathbf{y}_i^{r-1}$; $\quad \forall i = 1, \ldots, m$
  - Compute $\mathbf{y}_i^r = \sum_{j \in \mathcal{N}_i} w_{ij} \mathbf{y}_j^{r-1} + \nabla f_i(\mathbf{x}_i^r) - \nabla f_i(\mathbf{x}_i^{r-1})$; $\quad \forall i = 1, \ldots, m$
  - Exchange $\mathbf{x}_i^r$ and $\mathbf{y}_i^r$ with neighboring nodes; $\quad \forall i = 1, \ldots, m$
- **end for**
- **for** $j = 1 : \log(\frac{1}{\delta_1})$ **do**
  - Choose index $\tilde{t}_j \sim [0, T_1]$ uniformly at random and run Consensus Protocol on $\tilde{t}_j$ to find first order stationary point $\tilde{\mathbf{x}}$ with small gradient tracking disagreement;
- **end for**

Result: Returns first order stationary point $\tilde{\mathbf{x}}$ with probability at least $1 - \delta_1$

Algorithm 3: PDGT algorithm: Phase II

- **Input:** $\tilde{\mathbf{x}}, \eta_2, T_2, \mathcal{R}, B$
- All nodes sample a vector $\xi \sim$ uniform ball of radius $\mathcal{R}$ using the same seed;
- Set $\mathbf{x}_i^0 = \tilde{\mathbf{x}}_i + \xi$ and run Average Consensus on $\nabla f_i(\mathbf{x}_i^0)$ to set $\mathbf{y}_i^0 = \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(\mathbf{x}_i^0)$;
- **for** $r = 1, \ldots, T_2$ **do**
  - Compute $\mathbf{x}_i^r = \sum_{j \in \mathcal{N}_i} w_{ij} \mathbf{x}_j^{r-1} - \eta_2 \mathbf{y}_i^{r-1}$; $\quad \forall i = 1, \ldots, m$
  - Compute $\mathbf{y}_i^r = \sum_{j \in \mathcal{N}_i} w_{ij} \mathbf{y}_j^{r-1} + \nabla f_i(\mathbf{x}_i^r) - \nabla f_i(\mathbf{x}_i^{r-1})$; $\quad \forall i = 1, \ldots, m$
  - Exchange $\mathbf{x}_i^r$ and $\mathbf{y}_i^r$ with neighboring nodes; $\quad \forall i = 1, \ldots, m$
- **end for**
- Run Average Consensus Protocol for iterates $\underline{\mathbf{x}}^{T_2}$ and $\tilde{\mathbf{x}}$;
- **if** $H(\underline{\mathbf{x}}^{T_2}, \underline{\mathbf{y}}^{T_2}) - H(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) > -B$ **then**
  - Return approximate second-order stationary point $\tilde{\mathbf{x}} = [\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_m]$ and set $S = 1$;
- **else**
  - Return $\underline{\mathbf{x}}^{T_2} = [\mathbf{x}_1^{T_2}, \ldots, \mathbf{x}_m^{T_2}], \quad \underline{\mathbf{y}}^{T_2} = [\mathbf{y}_1^{T_2}, \ldots, \mathbf{y}_m^{T_2}]$ and set $S = 0$;
- **end if**

- Perturbed Decentralized Gradient Tracking (PDGT) consists of two phases. It runs the descent phase and escaping phase alternatively.
- The descent phase aims to find a first-order stationary point using decentralized gradient tracking.
- After drawing perturbations, the escaping phase is used to discriminate if the candidate point is a local minimum.

Restrictions of PDGT

- Deterministic gradient oracle. → Stochastic ?
- Fix number of iterations in the descent phase. → Adaptive ?
  - Stuck at saddle point for a long time.
  - Hard to be extended to stochastic gradient oracle.
- Consensus protocol over the entire network. → Independent ?
  - Compute $\bar{x}$ and $H(\bar{x})$ periodically.

Contributions

- We propose a novel PErturbed DEcentralized STorm ALgorithm (PEDESTAL), which is the first decentralized stochastic gradient-based algorithm to achieve second-order optimality with theoretical guarantees.
- We provide a new analysis framework to support changing phases on each worker node adaptively and independently.
- We prove that our method achieves (ϵ, ϵ_H) second-order stationary point with the complexity of $\tilde{O}(\epsilon^{-3})$, which matches the best results of decentralized algorithms to find first-order optimality or centralized algorithms to find second-order optimality.

PEDESTAL

Algorithm 1 Perturbed Decentralized STORM Algorithm (PEDESTAL)

Input: initial value $x_0^{(i)} = x_0$, $v_{-1}^{(i)} = \mathbf{0}$, $y_{-1}^{(i)} = \mathbf{0}$, $esc^{(i)} = -1$.
Parameter: $b_0, b_1, \eta, \beta, r, C_v, C_d, C_T$.

On i-th node:
for $t = 0, 1, \ldots, T-1$ **do**
  if $t = 0$ **then**
    Compute $v_0^{(i)} = \nabla F_i(x_0, \xi_0^{(i)})$ with $|\xi_0^{(i)}| = b_0$.
  else
    Compute $v_t^{(i)} = \nabla F_i(x_t^{(i)}, \xi_t^{(i)}) + (1-\beta)(v_{t-1}^{(i)} - \nabla F_i(x_{t-1}^{(i)}, \xi_t^{(i)}))$ with $|\xi_t^{(i)}| = b_1$.
  end if
  Communicate and update the gradient tracker: $y_t^{(i)} = \sum_{j=1}^{n} w_{ij}(y_{t-1}^{(j)} + v_t^{(j)} - v_{t-1}^{(j)})$.
  if $esc^{(i)} = -1$ and $\|y_t^{(i)}\| \leq C_v$ **then**
    Draw a perturbation $\xi \sim B_0(r)$ and update $z_t^{(i)} = x_t^{(i)} + \xi$.
    Save $x_t^{(i)}$ as $\tilde{x}^{(i)}$ and set $esc^{(i)} = 0$.
  else
    Update $z_t^{(i)} = x_t^{(i)} - \eta y_t^{(i)}$.
  end if
  Communicate and update the model parameter: $x_{t+1}^{(i)} = \sum_{j=1}^{n} w_{ij} z_t^{(j)}$.
  if $esc^{(i)} \geq 0$ **then**
    Reset $esc^{(i)} = -1$ **if** $\|x_{t+1}^{(i)} - \tilde{x}^{(i)}\| > C_d$ **else** update $esc^{(i)} = esc^{(i)} + 1$.
  end if
end for
Return: $\bar{x}_{t-C_T}$ if there are at least $\frac{n}{10}$ nodes satisfying $esc^{(i)} \geq C_T$.

- We adopt variance reduced gradient estimator.
- Parameter $esc^{(i)}$ represents the status on node i. It is -1 when the node is in the descent phase. Otherwise, it is the number of iterations that has been updated in the escaping phase.
- Each node can switch phase independently. The escaping phase is started according to real-time local gradient tracker. The escaping phase is broken if the moving distance of the model parameter is larger than a threshold C_d.
- The algorithm is terminated if at least 1/10 of total nodes satisfy $esc^{(i)} \geq C_T$.

Theorems

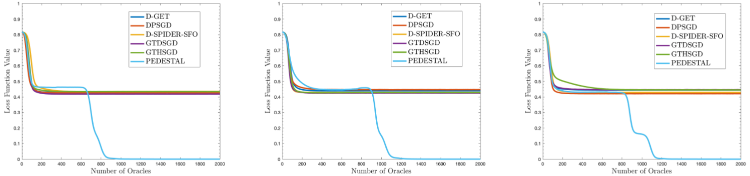
Let $\epsilon_H = \epsilon^\alpha$. When $\alpha \leq 0.5$, we have the following Theorem 1.

Theorem 1. *Assume $\alpha \leq 0.5$ and Assumption 1 to 5 are satisfied. Let $\theta = \min\{\frac{3-5\alpha}{2}, 1\}$. We set $\eta = \Theta(\frac{\epsilon^\theta}{L})$, $\beta = \Theta(\epsilon^{1+\theta})$, $b_0 = \Theta(\epsilon^{-2})$, $b_1 = \Theta(\max\{\epsilon^{2-\theta-5\alpha}, 1\})$, $r = \Theta(\epsilon^{1+\theta})$, $C_v = \Theta(\epsilon)$, $C_T = \tilde{\Theta}(\epsilon^{-\theta-\alpha})$ and $C_d = \tilde{\Theta}(\epsilon^{1-\alpha})$. Then our PEDESTAL algorithm will achieve $O(\epsilon, \epsilon_H)$-second-order stationary point with $\tilde{O}(\epsilon^{-3})$ gradient complexity.*

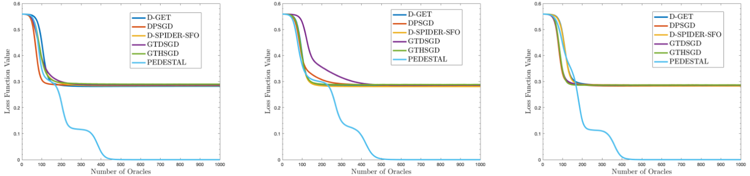
The specific constants hidden in $\Theta(\cdot)$ will be shown in Appendix B, where the proof outline and the completed proof of Theorem 1 can also be found. From Theorem 1 we can see our PEDESTAL-S with $b_1 = O(1)$ can achieve $O(\epsilon, \epsilon_H)$-second-order stationary point with $\tilde{O}(\epsilon^{-3})$ gradient complexity for $\epsilon_H \geq \epsilon^{0.2}$. In the classic setting, our PEDESTAL achieves second-order stationary point with $\tilde{O}(\epsilon^{-3})$ gradient complexity. When $\alpha > 0.5$, *i.e.*, $\epsilon_H < \sqrt{\epsilon}$, we have the following Theorem 2. Since the parameter settings are different and the $O(1)$ batchsize is only available in Theorem 1, we separate these two theorems. The proof of Theorem 2 can be found in Appendix D.

Theorem 2. *When $\epsilon_H < \sqrt{\epsilon}$ (i.e., $\alpha > 0.5$), we set $\eta = \tilde{\Theta}(\epsilon^\theta)$, $\beta = \Theta(\epsilon^{1+\theta})$, $b_0 = \Theta(\epsilon^{-1})$, $b_1 = \tilde{\Theta}(\epsilon^{-\max\{4\alpha-1-\theta, \theta+\alpha\}})$, $r = \Theta(\epsilon^{1+\theta})$, $C_v = \Theta(\epsilon)$, $C_T = \tilde{\Theta}(\epsilon^{-\theta-\alpha})$ and $C_d = \tilde{\Theta}(\epsilon^\alpha)$ where $\theta = \min\{\frac{3\alpha-1}{2}, 3\alpha - 2\}$. Under Assumption 1 to 5, our PEDESTAL algorithm will achieve $O(\epsilon, \epsilon_H)$-second-order stationary point with $\tilde{O}(\epsilon\epsilon_H^{-8} + \epsilon^4\epsilon_H^{-11})$ gradient complexity.*

Experiments

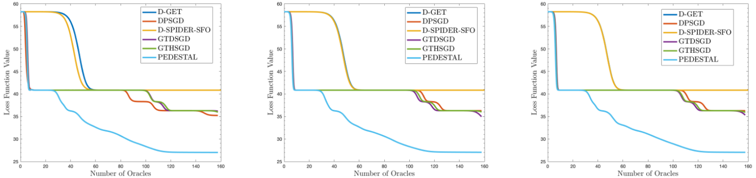


(a) $d = 50$, ring (b) $d = 50$, toroidal (c) $d = 50$, exponential

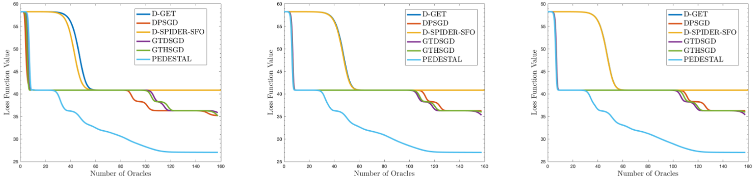


(d) $d = 100$, ring (e) $d = 100$, toroidal (f) $d = 100$, exponential

Figure 1: Experimental results of the decentralized matrix sensing task on different network topology for $d = 50$ and $d = 100$. Data is assigned to worker nodes by random distribution. The y-axis is the loss function value and the x-axis is the number of gradient oracles divided by the number of data N.



(a) ring, random (b) toroidal, random (c) exponential, random



(d) ring, Dirichlet (e) toroidal, Dirichlet (f) exponential, Dirichlet

Figure 2: Experimental results of the decentralized matrix factorization task on different network topology on MovieLens-100k. The y-axis is the loss function value and the x-axis is the number of gradient oracles divided by the size of matrix $N \times l$.

UNIVERSITY OF MARYLAND

Thank You!

UNIVERSITY OF MARYLAND