Multi-Agent Optimization for Distributed Learning

David Nardi

April 2025

1 Centralized problem

Given a dataset $\mathcal{D} = \big\{k \mid (\psi_k, y_k), k = 1, \dots, K\big\}$ where ψ_k is the vector of features (in this example we take $\psi_k = (x_{1k}, x_{2k}, 1) \in \mathbb{R}^p$) and y_k is a scalar value representing the ground-truth. The ground-truth model is assumed as

$$y_k = w^T\psi_k + \varepsilon_k, \quad \text{where} \quad \psi_k = (x_{1k}, \dots, x_{pk}) \text{ and } \varepsilon_k \sim \mathcal{N}(0, \sigma^2)$$

A single agent solves the learning problem with these steps:

1. $q = \sum_{k=1}^K \psi_k y_k \in \mathbb{R}^p$
2. $\Omega = \sum_{k=1}^K \psi_k \psi_k^T \in \mathbb{R}^{p\times p}$
3. $w^* = \Omega^{-1}q \in \mathbb{R}^p$ unique optimal solution

This has the assumption that $w \sim p(w) = \mathcal{N}(\mu, P)$ with $\mu = \Omega^{-1}q$ and $P = \Omega^{-1}$. We will exploit the Bayesian perspective for the consensus algorithm.

2 Distributed problem

The full dataset is split between each agent, i.e. each agent i has a fraction of the total indices $\mathcal{K}_i \subset \{1, \dots, K\}$. Hence, the dataset for each agent will be $\mathcal{K}_i = \big\{k \mid \big([\psi_k; \varphi_k^i], y_k\big)\big\}$, in this example we take $[\psi_k; \varphi_k^i] = (x_{1k}, x_{2k}, 1)$ that means each agent has a specific bias parameter. Each agent measures a specific feature, and we want to include this feature in the model too.

`agent=Agent()`

We assume the local model for the ground-truth for each agent i as follows

$$y_k^i = w^T\psi_k + \theta_i^T\varphi_k^i + \varepsilon_k, \quad \text{where} \quad \begin{matrix} \psi_k = (x_{1k}, \dots, x_{pk}) \\ \varphi_k^i = (x_{1k}^i, \dots, x_{p_ik}^i) \end{matrix} \quad \text{and} \quad \varepsilon_k \sim \mathcal{N}(0, \sigma^2)$$

`agent.features`
`agent.targets`

in this example we have $p = 2$ and $p_i = 1\ \forall i = 1, \dots, N$ with N being the number of agents in the network. Each agent i solves the problem locally as before:

1. $q_i = \sum_{k\in\mathcal{K}_i}[\psi_k; \phi_k^i]y_k \in \mathbb{R}^{p+p_i}$
2. $\Omega_i = \sum_{k\in\mathcal{K}_i}[\psi_k; \phi_k^i][\psi_k; \phi_k^i]^T \in \mathbb{R}^{(p+p_i)\times(p+p_i)}$
3. $w_i^* = \Omega_i^{-1}q_i \in \mathbb{R}^{p+p_i}$ where $w_i = [w; \theta_i]$

`agent.fit()`
`agent.w_i`

The assumption here for the prior is $w_i \sim p(w_i) = \mathcal{N}(\mu_i, P_i)$ where $\mu_i = \Omega_i^{-1}q_i$ and $P_i = \Omega_i^{-1}$ as before. This will be the starting point for the consensus algorithm that allows to align the local solutions fro each agent. The complexity is that each agent has a common part and an agent-specific part.

`agent.mu_i`
`agent.sigma_i`

Bayesian stuffs

We decompose $p(w_i) = \mathcal{N}(\mu_i, P_i)$ as follows exploiting the Bayesian interpretation for the prior

$$p(w_i) = p(w, \theta_i) = p(\theta_i|w)p(w) \quad \text{with} \quad \mu_i = \begin{bmatrix} \mu_{1i} \\ \mu_{2i} \end{bmatrix} \text{ and } P_i = \begin{bmatrix} P_{11i} & P_{12i} \\ P_{21i} & P_{22i} \end{bmatrix}$$

```
agent.mu_i
agent.sigma_i
```

Staying with the Gaussian prior we can obtain the exact form of the decomposed joint distribution

$$\begin{aligned} p(w) &\sim \mathcal{N}(\mu_{1i}, P_{11i}) & \mu_{2|1i} &= \mu_{2i} + P_{21i}P_{11i}^{-1}(w - \mu_{1i}) \\ p(\theta_i|w) &\sim \mathcal{N}(\mu_{2|1i}, P_{2|1i}) & P_{2|1i} &= P_{22i} - P_{21i}P_{11i}^{-1}P_{12i} \end{aligned}$$

Metropolis weights

Since we only want to exploit local informations that are available to all agents, we use the Metropolis weights defined as follows:

$$\pi_{ij} = \begin{cases} 1 - \sum_{j \in \mathcal{N}_i} \pi_{ij} & \text{if } j = i \\ \frac{1}{1+\max\{d_i, d_j\}} & \text{if } j \in \mathcal{N}_i \\ 0 & \text{otherwise} \end{cases}$$

```
agent.update_neigh
agent.neighbors
agent.degree
agent.metropolis
```

where $\mathcal{N}_i$ is the list of neighbors for the agent i.

Fusing the common and specific parts

Once we have the local solution for each agent, we may proceed with the consensus algorithm for the common part of the weights, starting with

$$q_{1i}(0) = P_{11i}^{-1}\mu_{1i} \quad \text{and} \quad \Omega_{1i}(0) = P_{11i}^{-1}$$

```
agent.fit()
```

We make L consensus steps, one at the time for each agent i, as follows

```
agent.consensus_ste
agent.sync()
```

1. $q_{1i}(l+1) = \pi_{ii}q_{1i}(l) + \sum_{j \in \mathcal{N}_i} \pi_{ij}q_{1j}(l)$
2. $\Omega_{1i}(l+1) = \pi_{ii}\Omega_{1i}(l) + \sum_{j \in \mathcal{N}_i} \pi_{ij}\Omega_{1j}(l)$
3. $w_{1i}(l+1) = \mu_{1i}(l+1) = \left[\Omega_{1i}(l+1)\right]^{-1} q_{1i}(l+1)$

```
agent._q_1i
agent._q_1i_next

agent._omega_1i
agent._omega_1i_nex

agent.w_i
agent.mu_i_next
agent.sigma_i_new
```

this will yield $q_{1i}(L)$, $\Omega_{1i}(L)$ and then $w_{1i}(L)$.
Eventually we can update the agent-specific parameters (just the bias here) having the common part already updated after consensus

$$\mu_{2i}(L) = \mu_{2i} + P_{21i}P_{11i}^{-1}\big(\mu_{1i}(L) - \mu_{1i}\big)$$

```
agent.local_consens
agent.mu_i_new
agent.sigma_i_new
```

where μ_{2i} is the mean from the local data distribution, this will update the distribution with informations from other agents in the network.