CONTINUAL LEARNING WITH DISTRIBUTED OPTIMIZATION: DOES COCOA FORGET?

Martin Hellkvist *Ayça Özçelikkale* *Anders Ahlén*

Dept. of Electrical Engineering, Uppsala University, Sweden

ABSTRACT

We focus on the continual learning problem where the tasks arrive sequentially and the aim is to perform well on the newly arrived task without performance degradation on the previously seen tasks. In contrast to the continual learning literature focusing on the centralized setting, we investigate the distributed estimation framework. We consider the well-established distributed learning algorithm CoCoA. We derive closed form expressions for the iterations for the overparametrized case. We illustrate the convergence and the error performance of the algorithm based on the over/underparametrization of the problem. Our results show that depending on the problem dimensions and data generation assumptions, CoCoA can perform continual learning over a sequence of tasks, i.e., it can learn a new task without forgetting previously learned tasks, with access only to one task at a time.

1. INTRODUCTION

When presented with a stream of data, continual learning [1,2] is the act of learning from new data while not forgetting what was learnt previously. New data can, for instance, come from a related classification task with new fine-grained classes, or it can have statistical distribution shift compared to the previously seen data. Each set of data that is presented to the model is referred to as a *task*. Continual learning aims to create models which perform well on all seen tasks without the need to retrain from scratch when new data comes [3–7].

The central issue in continual learning is *forgetting*, i.e., the drop in performance for old tasks as a new one is learned [1, 6, 7]. Forgetting is closely related to the error performance under non-stationary distributions and adaptation in estimation [8]. If a model performs poorly on old tasks, it is said to exhibit *catastrophic forgetting* [1, 6, 7]. To achieve continual learning, the forgetting must be reduced. Nevertheless, the key enabling factors for achieving continual learning are not well established, even for linear models [1,7].

We consider the continual learning problem from a distributed learning perspective, where optimization is performed over a network of computational nodes. In addition to supporting scalability, distributed learning is also attractive for the scenarios where data is already distributed over a network, for example in sensor networks [9, 10] and dictionary learning where sub-dictionaries are naturally separated over the network [11]. In this work, we focus on the well-established distributed learning algorithm CoCoA [12–15], which allows the nodes to use a local solver of their choice for their local subproblems.

We investigate the ability of CoCoA to perform continual learning for a linear model. We derive closed form expressions for the iterations of CoCoA when the number of unknown parameters exceeds the number of samples per task. We illustrate the performance of the algorithm with varying number of tasks and data points, revealing the trade-offs between the forgetting, the convergence performance and the dimensions of both the offline centralized problem, i.e., the problem of solving all tasks in one large batch in a centralized manner, and the nodes' local problems. **Our main findings** can be summarized as follows:

- CoCoA can exhibit continual learning by modifying the initialization point of the algorithm.
- CoCoA does not forget if all the tasks are underparametrized and have a shared solution.
- Even when each task comes only once, CoCoA can learn the new tasks with a reasonable forgetting performance. Whether the convergence and forgetting can be improved with task repetition depends on the data generation model.
- If the total number of samples over all tasks is close to the total number of parameters, then the algorithm may have slow convergence and relatively large error, even when there is a shared solution that solves all the tasks.

2. PROBLEM STATEMENT

2.1. Definition of a Task

The data consists of the matrix $\boldsymbol{A} \in \mathbb{R}^{n \times p}$ with the input vectors $\boldsymbol{a}_i^\intercal \in \mathbb{R}^{1 \times p}$ as rows, and a vector of corresponding outputs $\boldsymbol{y} = [y_1, \ldots, y_p]^\intercal \in \mathbb{R}^{n \times 1}$, where $y_i \in \mathbb{R}^{1 \times 1}$. We focus on fitting a linear model $f(\boldsymbol{a}_i)$ such that for each input-output pair $\boldsymbol{a}_i, y_i$, we have

$$y_i \approx f(\boldsymbol{a}_i) = \boldsymbol{a}_i^\intercal \boldsymbol{w}, \quad \forall i \tag{1}$$

where $\boldsymbol{w} \in \mathbb{R}^{p \times 1}$ is the vector of model parameters. We refer to the above regression problem with the data $(\boldsymbol{A}, \boldsymbol{y})$ as a *task*.

2.2. Continual Learning

We would like to fit the linear model in (1) to a sequence of tasks

$$\mathcal{S} = \left\{ (\boldsymbol{A}_{\tau(t)}, \boldsymbol{y}_{\tau(t)}) \right\}_{t=1}^{T}. \tag{2}$$

At a given time instant t, we have access to $(\boldsymbol{A}_{\tau(t)}, \boldsymbol{y}_{\tau(t)})$ but not to the other tasks. Hence, we would like to solve all tasks in $\mathcal{S}$ when we have access to only one of them at a given time. Throughout the paper, we apply the following assumption on the tasks $(\boldsymbol{A}_{\tau(t)}, \boldsymbol{y}_{\tau(t)})$,

Assumption 1. *There exists a solution $\boldsymbol{w}_\star \in \mathbb{R}^{p \times 1}$ that solves all the tasks in $\mathcal{S}$, i.e.,*

$$\exists\, \boldsymbol{w}_\star \in \mathbb{R}^{p \times 1} : \ \boldsymbol{A}_{\tau(t)} \boldsymbol{w}_\star = \boldsymbol{y}_{\tau(t)},\ t = 1, \ldots, T. \tag{3}$$

Assumption 1 constitutes a reasonable scenario in the case of highly overparameterized models and has been used in the centralized continual learning setting of [7]. We consider the investigation of this

Algorithm 1: Implementation of CoCoA [12] for (5).

Input: Task sequence $\mathcal{S} = \{(\boldsymbol{A}_{\tau(t)},\, \boldsymbol{y}_{\tau(t)})\}_{t=1}^{T}$, partitioning scheme $\mathcal{P} = \{p_k\}_{k=1}^{K}$, number of simulations T_c for CoCoA to run per task.
Initialize: $\boldsymbol{w}_0 = \boldsymbol{0}$.
for $t = 1, \ldots, T$ **do**
  $\boldsymbol{x}^0 = \boldsymbol{w}_{t-1}$
  $\boldsymbol{v}_k^0 = \boldsymbol{A}_{\tau(t),[k]}\boldsymbol{w}_{t-1,[k]}\ \forall\, k.$
  for $i = 0, 1, \ldots, T_c - 1$ **do**
    $\bar{\boldsymbol{v}}^i = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{v}_k^i$
    for $k \in \{1, 2, \ldots, K\}$ **do**
      $\Delta\boldsymbol{x}_{[k]}^i = \frac{1}{K}\boldsymbol{A}_{\tau(t),[k]}^{+}\left(\boldsymbol{y}_{\tau(t)} - \bar{\boldsymbol{v}}^i\right)$
      $\boldsymbol{x}_{[k]}^{i+1} = \boldsymbol{x}_{[k]}^i + \Delta\boldsymbol{x}_{[k]}^i$
      $\boldsymbol{v}_k^{i+1} = \bar{\boldsymbol{v}}^i + K\boldsymbol{A}_{\tau(t),[k]}\Delta\boldsymbol{x}_{[k]}^i$
  $\boldsymbol{w}_t = \boldsymbol{x}^{T_c}$

setting as good starting point for the general case since it facilitates a tractable analysis. However, it is not always natural to make this assumption, hence we study a setting where Assumption 1 does not hold in Section 5.

For a given parameter vector $\boldsymbol{w}$, we measure the forgetting by the squared loss on a specific task $(\boldsymbol{A}_m,\, \boldsymbol{y}_m)$, i.e., $\frac{1}{n_m}\left\|\boldsymbol{A}_m\boldsymbol{w} - \boldsymbol{y}_m\right\|^2$, as is done in previous works [7,16]. While training over the sequence of tasks $\mathcal{S}$, a corresponding sequence of parameter vectors estimates is generated, denoted by $\boldsymbol{w}_t, t = 1, \ldots, T$. We denote the forgetting of $\boldsymbol{w}_t$ as the forgetting of the tasks up until and including the current task,

$$\mathcal{F}_{\mathcal{S}}(t) = \frac{1}{t}\sum_{i=1}^{t}\frac{1}{n_{\tau(i)}}\left\|\boldsymbol{A}_{\tau(i)}\boldsymbol{w}_t - \boldsymbol{y}_{\tau(i)}\right\|^2. \tag{4}$$

Note that forgetting is defined in terms of data fit for $\boldsymbol{y}_{\tau(i)}$ instead of estimation accuracy for $\boldsymbol{w}_\star$.

2.3. Distributed Continual Learning with CoCoA

In CoCoA, the unknowns are distributed over a network of K nodes [12–15]. The unknown vector $\boldsymbol{w}$ is partitioned according to the partitioning $\mathcal{P} = \{p_k\}_{k=1}^{K}$, such that node k governs p_k of the p unknowns in $\boldsymbol{w} \in \mathbb{R}^{p\times 1}$. Note that this is in contrast to settings where the observations are distributed over the network. The partitioning consists of mutually exclusive sets of indices, hence each entry of the unknown vector (hence each column of a given feature matrix) is associated with only one node, i.e., $\sum_{k=1}^{K} p_k = p$.

For each task $(\boldsymbol{A}_m,\, \boldsymbol{y}_m)$, the columns of $\boldsymbol{A}_m \in \mathbb{R}^{n_m\times p}$ is partitioned according to the partitioning for the unknowns. We denote the matrix of columns associated with node k as $\boldsymbol{A}_{m,[k]}$. For each presented task $\tau(t)$ in the sequence $\mathcal{S}$, we run CoCoA to minimize the forgetting of the current task, i.e., $\min_{\boldsymbol{w}_t}\left\|\boldsymbol{A}_{\tau(t)}\boldsymbol{w}_t - \boldsymbol{y}_{\tau(t)}\right\|^2$. CoCoA is an iterative algorithm, which is run for T_c iterations for each task. For notational clarity, the unknown vector is represented by $\boldsymbol{w}_t$ for the outer iteration t over the tasks and by $\boldsymbol{x}_i$ for the inner CoCoA iteration i. The corresponding partitionings of $\boldsymbol{w}_t$ and $\boldsymbol{x}_i$ are denoted by $\boldsymbol{w}_{t,[k]}$ and $\boldsymbol{x}_{i,[k]}$, respectively. Hence the overall training scheme, presented in Algorithm 1, consists of three nested for-loops: the *outer iterations* $t = 1, \ldots, T$ over the sequence of tasks; the *inner iterations* $i = 0, \ldots, T_c - 1$ of CoCoA; and the parallelization over the nodes $k = 1, \ldots, K$.

For task $\tau(t)$, the local subproblems for the nodes to solve in iteration i of CoCoA are then [12, Sec. 3.1]

$$\begin{aligned}\min_{\Delta\boldsymbol{x}_{[k]}^i}\frac{1}{2Kn_{\tau(t)}}\left\|\bar{\boldsymbol{v}}^i - \boldsymbol{y}_{\tau(t)}\right\|^2 + \frac{\sigma'}{2n_{\tau(t)}}\left\|\boldsymbol{A}_{\tau(t),[k]}\Delta\boldsymbol{x}_{[k]}^i\right\|^2 \\ + \frac{1}{n_{\tau(t)}}\left(\bar{\boldsymbol{v}}^i - \boldsymbol{y}_{\tau(t)}\right)^{\intercal}\boldsymbol{A}_{\tau(t),[k]}\Delta\boldsymbol{x}_{[k]}^i.\end{aligned} \tag{5}$$

This is a convex problem in $\Delta\boldsymbol{x}_{[k]}^i$. Hence, we set the gradient w.r.t. $\Delta\boldsymbol{x}_{[k]}^i$ to zero to obtain the minimum ℓ_2-norm solution as $\Delta\boldsymbol{x}_{[k]}^i = \frac{1}{\sigma'}\boldsymbol{A}_{\tau(t),[k]}^{+}\left(\boldsymbol{y}_{\tau(t)} - \bar{\boldsymbol{v}}^i\right)$, where $(\cdot)^+$ denotes the Moore-Penrose pseudoinverse. Using the notation of [12], we set $\sigma' = \gamma K$, with $\gamma \in (0, 1]$, as these are considered safe choices [15]. With these choices, these parameters cancel out to give the explicit expressions in Algorithm 1.

In order to utilize the knowledge from the previous tasks, we do the following: i) we initialize CoCoA with the parameter vector from the previous step, i.e., we set $\boldsymbol{x}^0 = \boldsymbol{w}_{t-1}$; ii) we set the corresponding initial point for the local contributions as $\boldsymbol{v}_k^0 = \boldsymbol{A}_{\tau(t),[k]}\boldsymbol{x}_{[k]}^0$.

Throughout the paper, we apply the following assumption on the local partitions of the tasks

Assumption 2. *The matrices $\boldsymbol{A}_{\tau(t),[k]} \in \mathbb{R}^{n_{\tau(t)}\times p_k}$ are full rank.*

2.4. Offline and centralized problem

Throughout our analysis of the distributed continual learning problem, we consider the *offline and centralized problem* as the reference problem, which corresponds to solving all tasks in $\mathcal{S}$ simultaneously in a centralized fashion, i.e.,

$$\boldsymbol{y}_{\mathcal{S}} = \boldsymbol{A}_{\mathcal{S}}\boldsymbol{w},\ \boldsymbol{y}_{\mathcal{S}} = \begin{bmatrix}\boldsymbol{y}_1\\ \vdots \\ \boldsymbol{y}_M\end{bmatrix} \in \mathbb{R}^{N\times 1},\ \boldsymbol{A}_{\mathcal{S}} = \begin{bmatrix}\boldsymbol{A}_1\\ \vdots \\ \boldsymbol{A}_M\end{bmatrix} \in \mathbb{R}^{N\times p}, \tag{6}$$

where we denote the unique set of tasks in $\mathcal{S}$ by $\{(\boldsymbol{A}_m,\, \boldsymbol{y}_m),\, m = 1, \ldots, M\}$, and the total number of samples over all the unique tasks by $N = \sum_{m=1}^{M} n_m$.

3. OVERPARAMETERIZED LOCAL PROBLEMS

If $p_k \geq n_m$ for all $k = 1, \ldots, K$ and $m = \tau(1), \ldots, \tau(T)$, then the local problems are overparameterized, i.e., the nodes' local feature matrices $\boldsymbol{A}_{\tau(t),[k]} \in \mathbb{R}^{n_{\tau(t)}\times p_k}$ are broad. By Assumption 2, the matrices $\boldsymbol{A}_{\tau(t),[k]}$ have rank $n_{\tau(t)}$, thus

$$\boldsymbol{A}_{\tau(t),[k]}\boldsymbol{A}_{\tau(t),[k]}^{+} = \boldsymbol{I}_{n_{\tau(t)}}. \tag{7}$$

Within CoCoA, the updates of $\boldsymbol{v}_k^{i+1}$ for $i \geq 0$ (line 11 in Algorithm 1) can be rewritten as

$$\boldsymbol{v}_k^{i+1} = \bar{\boldsymbol{v}}^i + \boldsymbol{A}_{\tau(t),[k]}\boldsymbol{A}_{\tau(t),[k]}^{+}\left(\boldsymbol{y}_{\tau(t)} - \bar{\boldsymbol{v}}^i\right) = \boldsymbol{y}_{\tau(t)}. \tag{8}$$

It follows that $\bar{\boldsymbol{v}}^i = \frac{1}{K}\sum_{k=1}^{K}\boldsymbol{v}_k^i = \boldsymbol{y}_{\tau(t)},\ i \geq 1$ and as a result, the steps $\Delta\boldsymbol{x}_{[k]}^i$ for $i \geq 1$ are zero, i.e.,

$$\Delta\boldsymbol{x}_{[k]}^i = \frac{1}{K}\boldsymbol{A}_{\tau(t),[k]}^{+}\left(\boldsymbol{y}_{\tau(t)} - \bar{\boldsymbol{v}}^i\right) = \boldsymbol{0}. \tag{9}$$

In other words, CoCoA in this setting converges after its first iteration $i = 0$. To find the solution $\boldsymbol{x}^1$ that CoCoA converges to, we

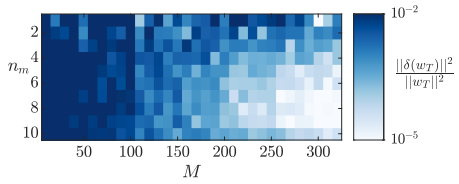

Fig. 1: The size of the final step from $\boldsymbol{w}_{T-1}$ to $\boldsymbol{w}_T$ after training on each task once, versus the number of tasks M and the number of samples per task n_m.

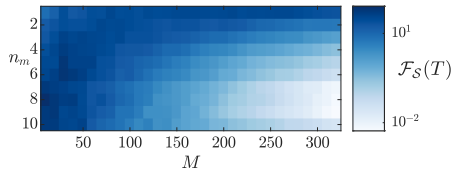

Fig. 2: The forgetting of $\boldsymbol{w}_T$ after training on each task once, versus the number of tasks M and the number of samples per task n_m.

insert the initial values of $\boldsymbol{x}^0 = \boldsymbol{w}_{t-1}$, and $\boldsymbol{v}_k^0 = \boldsymbol{A}_{\tau(t),[k]}\boldsymbol{w}_{t-1,[k]}$, finding that

$$\bar{\boldsymbol{v}}^0 = \frac{1}{K}\boldsymbol{A}_{\tau(t)}\boldsymbol{w}_{t-1}, \tag{10}$$

and that the local estimates $\boldsymbol{x}_k^i$ in CoCoA converge to

$$\boldsymbol{x}_{[k]}^i = \boldsymbol{w}_{t-1,[k]} + \frac{1}{K}\boldsymbol{A}_{\tau(t),[k]}^{+}\left(\boldsymbol{y}_{\tau(t)} - \boldsymbol{A}_{\tau(t)}\boldsymbol{w}_{t-1}\right). \tag{11}$$

Combining over the nodes $k = 1, \ldots, K$, the parameter vector $\boldsymbol{w}_t = \boldsymbol{x}^1$ after training on task $\tau(t)$ is,

$$\boldsymbol{w}_t = \boldsymbol{P}_{\tau(t)}\boldsymbol{w}_{t-1} + \bar{\boldsymbol{A}}_{\tau(t)}\boldsymbol{y}_{\tau(t)}, \tag{12}$$

where we have introduced the notation

$$\boldsymbol{P}_{\tau(t)} = \left(\boldsymbol{I}_p - \bar{\boldsymbol{A}}_{\tau(t)}\boldsymbol{A}_{\tau(t)}\right), \tag{13}$$

$$\bar{\boldsymbol{A}}_{\tau(t)} = \frac{1}{K}\left[(\boldsymbol{A}_{\tau(t),[1]}^{+})^{\intercal}, \cdots, (\boldsymbol{A}_{\tau(t),[K]}^{+})^{\intercal}\right]^{\intercal} \in \mathbb{R}^{p \times n_{\tau(t)}}. \tag{14}$$

In this setting, CoCoA always solves the latest seen task, i.e.,

$$\hat{\boldsymbol{y}}_{\tau(t)} = \boldsymbol{A}_{\tau(t)}\boldsymbol{w}_t \tag{15}$$

$$= \boldsymbol{A}_{\tau(t)}\boldsymbol{P}_{\tau(t)}\boldsymbol{w}_{t-1} + \boldsymbol{A}_{\tau(t)}\bar{\boldsymbol{A}}_{\tau(t)}\boldsymbol{y}_{\tau(t)} \tag{16}$$

$$= \left(\boldsymbol{A}_{\tau(t)} - \boldsymbol{A}_{\tau(t)}\right)\boldsymbol{w}_{t-1} + \boldsymbol{y}_{\tau(t)} = \boldsymbol{y}_{\tau(t)}, \tag{17}$$

where we have used that $\boldsymbol{A}_{\tau(t)}\bar{\boldsymbol{A}}_{\tau(t)} = \boldsymbol{I}_{n_{\tau(t)}}$ as a result of (7). These findings are summarized in the following lemma.

Lemma 1. *If a feature matrix $\boldsymbol{A}_{\tau(t)}$ is full rank, and $p_k \geq n_{\tau(t)}$ for all k, then* CoCoA *(lines 4 – 11 in Algorithm 1) converges after the first iteration $i = 0$ when run on the task $(\boldsymbol{A}_{\tau(t)}, \boldsymbol{y}_{\tau(t)})$, and the resulting solution $\boldsymbol{w}_t$ in* (12) *solves the task, i.e., $\hat{\boldsymbol{y}}_{\tau(t)} = \boldsymbol{A}_{\tau(t)}\boldsymbol{w}_t = \boldsymbol{y}_{\tau(t)}$.*

Although the algorithm after each iteration t always solves the latest task, these solutions do not necessarily perform well on previously seen tasks. We now continue with investigating the convergence and forgetting dynamics of $\boldsymbol{w}_t$ in (12).

3.1. One shot at each task

We now consider a setting where every task only comes once in the sequence $\mathcal{S}$. In this section, we discuss numerical results from simulations where we generate the tasks as $\boldsymbol{y}_m = \boldsymbol{A}_m\boldsymbol{w}_\star$, $m = 1, \ldots, M$, where $\boldsymbol{A}_m \in \mathbb{R}^{n_m \times p}$ have independently generated standard Gaussian entries. We fix the number of parameters $p = 160$ and vary the number of samples per task n_m. We set the generating solution to $\boldsymbol{w}_\star = \mathbf{1}$. We set the training sequence as $\tau(t) = t$, $t = 1, \ldots, T$ with $T = M$, thus we train the model on each task once. We measure the convergence as the relative size of the last step taken by the outer iterations, i.e.,

$$\frac{\|\boldsymbol{\delta}(\boldsymbol{w}_T)\|^2}{\|\boldsymbol{w}_T\|^2}, \tag{18}$$

with $\boldsymbol{\delta}(\boldsymbol{w}_T) = \|\boldsymbol{w}_T - \boldsymbol{w}_{T-1}\|^2$. We fix the number of nodes in the network as $K = 4$, which together perform the CoCoA iterations, and the partitioning as $\mathcal{P} = \{p_k\}_{k=1}^K = \{16, 32, 48, 64\}$. Note that we only study $n_m \leq 10$, since if $n_m \approx p_k$ for any local problem, then CoCoA obtains extremely high error values with high probability [17]. We plot the convergence in Figure 1 and the forgetting in the final step $\mathcal{F}_{\mathcal{S}}(T)$ in Figure 2. We observe the following:

Remark 1. *Even though* CoCoA*, i.e., the inner iterations of Algorithm 1, converge in one step and solves the latest seen task (see Lemma 1), the obtained solution $\boldsymbol{w}_T$ after seeing all tasks once can exhibit forgetting if the number of tasks M or the number of data points per task n_m is too small. The forgetting can be improved by increasing the number of tasks M (hence increasing the number of data points) due to existence of the solution $\boldsymbol{w}_\star$ that solves all tasks.*

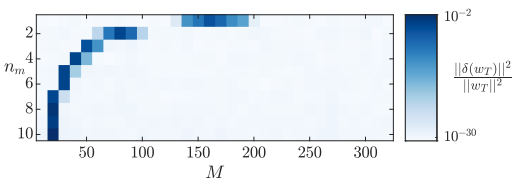

Fig. 3: The size of the final step from $\boldsymbol{w}_{T-1}$ to $\boldsymbol{w}_T$ after training on each task 1000 times, versus the number of tasks M and the number of samples per task n_m.

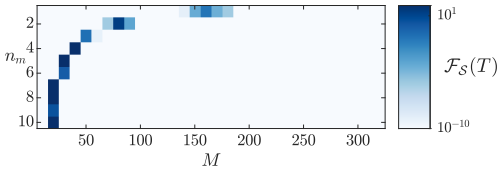

Fig. 4: The forgetting of $\boldsymbol{w}_T$ after training on each task 1000 times, versus the number of tasks M and the number of samples per task n_m.

3.2. Cyclic sequence of tasks

We now repeat the experiment of Section 3.1, but extend the training sequence so that the model trains on each task 1000 times. In other words, with M tasks we set the sequence as $\tau(t) = 1, \ldots, M, 1, \ldots, M, \ldots, t = 1, \ldots, T$, with $T = 1000M$. We plot the convergence in Figure 3 and the forgetting in Figure 4. The plots illustrate that the algorithm converges for most pairs of (n_m, M), and that the forgetting for the final solution is low. However, the plots also illustrate that both the convergence and forgetting is worse along a streak of pairs (n_m, M), for which $N = \sum_{m=1}^{M} n_m = p$, i.e., where the total number of samples N equals the number of parameters p, for example $(1, 160)$, $(2, 80)$ and $(4, 40)$. We now analyze this phenomenon, distinguishing between the following scenarios: *i)* $N > p$; *ii)* $N \approx p$; and *iii)* $N < p$.

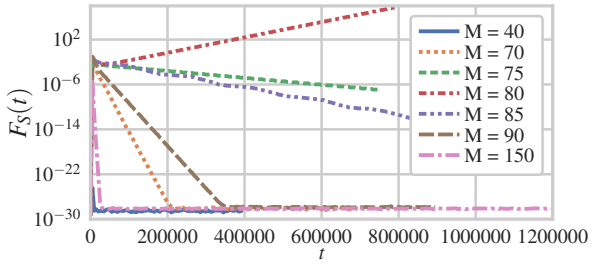

Fig. 5: The forgetting of $\boldsymbol{w}_t$ versus t for different number of tasks M with $n_m = 2$ samples each.

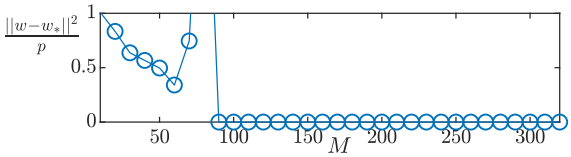

Fig. 6: The distance from $\boldsymbol{w}_T$ to the solution $\boldsymbol{w}_\star = \mathbf{1}$ used to generate the data y_m. Here, $n_m = 2$ for each task.

In the below, recall that n_m is constant over $m = 1, \ldots, M$, hence $N = Mn_m$. Recall also that as a result of how we generate the regressor matrices $\boldsymbol{A}_{\tau(t)}$ here (independent standard Gaussian matrices), the offline and centralized matrix $\boldsymbol{A}_{\mathcal{S}}$ has full rank with probability one.

i) $N > p$: If M or n_m is large enough so that the total number of samples $N = Mn_m$ exceeds the number of parameters p, then the offline and centralized problem $\boldsymbol{y}_{\mathcal{S}} = \boldsymbol{A}_{\mathcal{S}}\boldsymbol{w}_\star$ in (6) is underparameterized, i.e., there are less parameters than samples. The matrix $\boldsymbol{A}_{\mathcal{S}} \in \mathbb{R}^{N\times p}$ has full rank, and $\boldsymbol{y}_{\mathcal{S}} = \boldsymbol{A}_{\mathcal{S}}\boldsymbol{w}_\star$. Hence, $\boldsymbol{w}_\star$ is the unique solution to the offline and centralized problem. This suggests $\boldsymbol{w}_T = \boldsymbol{w}_\star$, since the solutions for large M or n_m in Figure 3 and 4 converge to a solution $\boldsymbol{w}_T$ which solves all tasks, $\boldsymbol{w}_\star$ is the only solution which does so.

ii) $N \approx p$: If the total number of samples $N = Mn_m$ is close or to the number of parameters p, then the offline and centralized feature matrix $\boldsymbol{A}_{\mathcal{S}}$ is nearly square or square. The results in Figure 3 and 4 illustrate that Algorithm 1 does not converge, and does not solve the previously seen tasks. The non-convergence of the algorithm for these pairs of (n_m, M) are further illustrated in Figure 5, where we plot the forgetting as a function of the outer iterations t in Algorithm 1, i.e., $\mathcal{F}_{\mathcal{S}}(t)$, where we have evaluated the forgetting every 1000 iterations, and fixed the number of samples per task to $n_m = 2$. Figure 5 illustrates that if $M = 80$, then the vectors $\boldsymbol{w}_t$ do not converge, and are instead diverging. If $M \approx 80$ but $M \neq 80$, then the convergence is very slow compared to if M is further away from $M = 80$.

iii) $N < p$: In this scenario, the offline and centralized problem is overparameterized. The feature matrix $\boldsymbol{A}_{\mathcal{S}}$ is full rank, hence there is possibly an infinite number of solutions $\boldsymbol{w}$ which solve all the tasks simultaneously. In this regime, $\boldsymbol{w}_t$ generally do not converge to $\boldsymbol{w}_\star$. In Figure 6, and plot the distance from $\boldsymbol{w}_T$ to the solution $\boldsymbol{w}_\star$ versus the number of tasks M. The results are from the same experiment as in Figure 3 and 4. Figure 6 illustrates that for small M, even though the final solution $\boldsymbol{w}_T$ solves all the tasks, it is not equal to $\boldsymbol{w}_\star = \mathbf{1}$. Consistent with our observations for scenario *i)*, Figure 6 suggests that $\boldsymbol{w}_t$ converges to $\boldsymbol{w}_\star$ for large M.

4. UNDERPARAMETERIZED TASK

A task $(\boldsymbol{A}_m, \boldsymbol{y}_m)$ such that $p < n_m$ is referred to as *underparameterized*, as there are less tuneable parameters in $\boldsymbol{w} \in \mathbb{R}^{p\times 1}$ than there are equations in $\boldsymbol{A}_m\boldsymbol{w} = \boldsymbol{y}_m$. It follows from Assumption 1 and 2 that $\boldsymbol{w}_\star$ is the unique solution to any underparameterized task $(\boldsymbol{A}_m, \boldsymbol{y}_m)$. By [12, Theorem 2], a sufficiently large number of iterations T_c of CoCoA guarantees an arbitrarily small optimality gap. In other words, if T_c is large enough while learning an underparameterized task $(\boldsymbol{A}_m, \boldsymbol{y}_m)$, then the estimate $\boldsymbol{x}^{T_c}$ will converge to $\boldsymbol{w}_\star$. Hence, we have the following lemma:

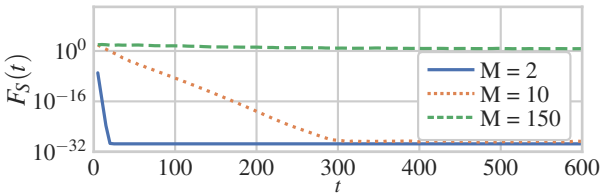

Fig. 7: The forgetting of $\boldsymbol{w}_t$ versus t when the task output vectors $\boldsymbol{y}_m$ are generated either by $\boldsymbol{w}_{even}$ or $\boldsymbol{w}_{odd}$.

Lemma 2. *If all tasks in the sequence $\mathcal{S}$ are underparameterized, and $T_c \to \infty$, then Algorithm 1 converges to $\boldsymbol{w}_\star$, which is the unique solver of all tasks in $\mathcal{S}$.*

5. FAMILY OF TASKS WITHOUT ASSUMPTION 1

We now consider a setting where Assumption 1 does not hold. We again consider the cyclic manner of training on M tasks as in Section 3.2. Tasks $(\boldsymbol{A}_m, \boldsymbol{y}_m)$ with even and odd indices m are generated with $\boldsymbol{y}_m = \boldsymbol{A}_m\boldsymbol{w}_{even}$, and $\boldsymbol{y}_m = \boldsymbol{A}_m\boldsymbol{w}_{odd}$, respectively. We set $\boldsymbol{w}_{even} = \mathbf{1}_{p\times 1}$, and $\boldsymbol{w}_{odd} = [\mathbf{1}_{1\times 0.9p}, \mathbf{0}_{1\times 0.1p}]^\intercal$. We randomly create the features matrices $\boldsymbol{A}_m$ as independent standard Gaussian matrices. We set $n_m = 2$ samples and vary the number of tasks M.

In Figure 7, we present the forgetting $\mathcal{F}_{\mathcal{S}}(t)$ versus the number of outer iterations t of Algorithm 1, for different values of M. These plots illustrate the different convergence behaviour for small and large M. In particular, for small values of M ($M = 2$), the algorithm quickly converges to a solution which solves all the tasks. (Although $\boldsymbol{w}_{even} \neq \boldsymbol{w}_{odd}$, there can be such a shared solution since the centralized, offline system is underparametrized for M small.) For larger values of M ($M = 10$), convergence is slower. For higher values of M ($M = 150$), the algorithm does not converge. Although not visible in Figure 7, the forgetting for $M = 150$ fluctuates on the range $[4, 7]$ in the plot for $t > 2000$.

These results illustrate that although the total number of samples for the two types of task with $\boldsymbol{w}_{even}$ and $\boldsymbol{w}_{odd}$ increases with M, the forgetting does not necessarily improve. This is consistent with the fact that for larger values of M there is a higher number of data points that needs to be satisfied by the solution vector of CoCoA.

6. CONCLUSIONS

We have applied the distributed learning algorithm CoCoA in a continual learning setting, and investigated its properties in terms of convergence and forgetting. Our results illustrate that CoCoA can be suitable for performing continual learning in a distributed manner, and that the forgetting can be reasonably low even if each task only comes once in the sequence. We have illustrated how the dynamics of the convergence and forgetting of the algorithm varies with the dimensions of both the local problems at the nodes as well as the offline and centralized problem. There are several interesting lines of future work, including characterization of convergence and forgetting behaviour under other data models as well as development of methods to improve the forgetting behaviour of the algorithm.

7. REFERENCES

[1] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proc. of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, mar 2017.

[2] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019.

[3] S. Legg and M. Hutter, "Universal intelligence: A definition of machine intelligence," *Minds & Machines*, vol. 17, pp. 391–444, 2007.

[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *Int. J. of Comp. Vision*, vol. 2015, pp. 211–252, 2015.

[5] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[6] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[7] I. Evron, E. Moroshko, R. Ward, N. Srebro, and D. Soudry, "How catastrophic can catastrophic forgetting be in linear regression?" in *Proc. of Thirty Fifth Conf. on Learning Theory*, vol. 178. PMLR, Jul 2022, pp. 4028–4079.

[8] A. H. Sayed, "Adaptation, learning, and optimization over networks," *Foundations and Trends in Machine Learning*, vol. 7, pp. 311–801, 2014.

[9] M. Rabbat and R. Nowak, "Distributed optimization in sensor networks," *Proc. 3rd Int. Symp. on Inf. Proc. in Sensor Netw.*, pp. 20–27, 2004.

[10] S. Kar and J. M. F. Moura, "Distributed consensus algorithms in sensor networks with imperfect communication: Link failures and channel noise," *IEEE Transactions on Signal Processing*, vol. 57, no. 1, pp. 355–369, 2009.

[11] J. Chen, Z. J. Towfic, and A. H. Sayed, "Dictionary learning over distributed models," *IEEE Trans. on Signal Process.*, vol. 63, no. 4, pp. 1001–1016, 2015.

[12] V. Smith, S. Forte, C. Ma, M. Takáč, M. I. Jordan, and M. Jaggi, "CoCoA: A general framework for communication-efficient distributed optimization," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 8590–8638, 2017.

[13] M. Jaggi, V. Smith, M. Takác, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan, "Communication-efficient distributed dual coordinate ascent," *Adv. Neural Inf. Process. Systems*, pp. 3068–3076, 2014.

[14] C. Ma, J. Konečnỳ, M. Jaggi, V. Smith, M. I. Jordan, P. Richtárik, and M. Takáč, "Distributed optimization with arbitrary local solvers," *Optimization Methods and Softw.*, vol. 32, no. 4, pp. 813–848, 2017.

[15] L. He, A. Bian, and M. Jaggi, "Cola: Decentralized linear learning," *Adv. Neural Inf. Process. Syst.*, pp. 4536–4546, 2018.

[16] T. Doan, M. Abbana Bennani, B. Mazoure, G. Rabusseau, and P. Alquier, "A theoretical analysis of catastrophic forgetting through the ntk overlap matrix," ser. Proceedings of Machine Learning Research, A. Banerjee and K. Fukumizu, Eds., vol. 130, 2021, pp. 1072–1080.

[17] M. Hellkvist, A. Özçelikkale, and A. Ahlén, "Linear regression with distributed learning: A generalization error perspective," *IEEE Trans. on Signal Process.*, vol. 69, pp. 5479–5495, 2021.