Asynchronous Decentralized Online Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Most existing algorithms in decentralized online learning are conducted in the synchronous setting. However, synchronization makes these algorithms suffer from the straggler problem, i.e., fast learners have to wait for slow learners, which significantly reduces such algorithms' overall efficiency. To overcome this problem, we study decentralized online learning in the asynchronous setting, which allows different learners to work at their own pace. We first formulate the framework of Asynchronous Decentralized Online Convex Optimization, which specifies the whole process of asynchronous decentralized online learning using a sophisticated event indexing system. Then we propose the Asynchronous Decentralized Online Gradient-Push (AD-OGP) algorithm, which performs asymmetric gossiping communication and instantaneous model averaging. We further derive a regret bound of AD-OGP, which is a function of the network topology, the levels of processing delays, and the levels of communication delays. Extensive experiments show that AD-OGP runs significantly faster than its synchronous counterpart and also verify the theoretical results.

1 Introduction

Decentralized online learning has recently received increasing research attention due to its capability of processing large-scale streaming data in decentralized computational architectures (e.g., mobile networks and CPU/GPU clusters). Many different algorithms have been proposed [31, 33, 22, 15, 32, 28, 16, 24] and successfully applied in various application scenarios, such as recommendation systems [29], energy management [5] and cooperative navigation [21].

Most of the existing decentralized online algorithms are investigated under the synchronous setting, where all the learners are restricted to communicating their updated information (e.g., model parameters, gradients, dual variables) with each other at the same pace. However, this synchronization mechanism incurs the straggler problem, which can markedly deteriorate the efficiency of the algorithms [30, 17, 3]. In more detail, at each iteration, the fast learners have to wait for the slow learners, i.e., the stragglers, to finish their update and communication, and consequently, their speeds will be dragged down by those of the stragglers; this may result in a significant reduction in the algorithm's overall efficiency as the speeds of different learners in a decentralized system usually vary widely owing to varying computing powers, storage capacities, network bandwidths, etc.

As is well acknowledged, designing the asynchronous counterpart of a synchronous algorithm is the most common way to resolve the straggler problem. However, compared to the well-studied centralized online [8, 18] and decentralized stochastic [17, 12, 3, 23] settings, how to do asynchronous learning in the decentralized online setting remains largely unexplored. The major obstacle is its intrinsic hardness. In asynchronous decentralized online learning, each learner has five basic types of

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

actions, namely prediction, weight update, message sending, message receiving, and model averaging. Due to the asynchronization mechanism, each learner executes its own actions independently. Hence the orders of the actions of different learners are intermixed together, which results in very complex interaction dynamics. Indeed currently, there does not even exist a formal framework that can accurately describe the whole process of asynchronous decentralized online learning, not to mention algorithm design or theoretical analysis.

In this paper, we conduct a systematic study of asynchronous decentralized online learning. To begin with, we formulate the framework of Asynchronous Decentralized Online Convex Optimization (AD-OCO). We first specify the action sequence of each learner using its own indices, then propose a novel event indexing system, which delicately characterizes the complete orders of actions of all learners by mapping them into a single time axis. To the best of our knowledge, this is the first formal framework that gives a complete characterization of the whole interaction dynamics of asynchronous decentralized online learning.

Based on the proposed AD-OCO framework, we develop the Asynchronous Decentralized Online Gradient-Push (AD-OGP) algorithm. AD-OGP consists of two key modules. The first module is a gossip-based communication scheme, in which each learner is only allowed to communicate with its immediate neighbors. In general, there are two major types of gossiping strategies in decentralized algorithms, namely symmetric gossiping [13, 17] and asymmetric gossiping [19, 3]. In *asymmetric gossiping*, the gossiping sender does not need to wait for the response message from the gossiping receiver after it sends out its message. Hence asymmetric gossiping does not require any local synchronization among learners, which is needed in symmetric gossiping. This makes asymmetric gossiping more suitable for asynchronous online learning. In particular, we adopt push-sum strategy [14, 19], which is the most common way to realize asymmetric gossiping. The second module is a local update scheme, which builds upon online gradient descent (OGD) [34]. When combining push-sum with OGD, if we directly apply the standard projection of OGD, the resulting algorithm crashes as it may generate predictions outside of the feasible region. To resolve this problem, we devise a novel *weighted projection* operation in OGD, which makes its local update compatible with push-sum. Moreover, in AD-OGP, we allow each learner to perform model averaging as long as it is unoccupied and its receiving buffer is not empty, which reduces communication delays to the most extent. We emphasize that, the above mechanisms, namely weighted projection and *instantaneous model averaging*, are two special designs when tailoring asynchronization to the online setting.

We further conduct the first regret analysis of asynchronous decentralized online learning. In particular, we extend the graph augmentation technique [10] to the online setting to handle message delays, then disentangle the complex effects of predictions, weight update, and model averaging via our proposed event indexing system. We derive a non-trivial regret bound of AD-OGP, which captures the effect of network size, network topology, levels of processing delays, and levels of message delays. Our bound is in the same order $O(\sqrt{T})$ as that of its synchronous counterpart [27]. In addition, we make an extra contribution to the convergence analysis of push-sum. Specifically, we reduce a factor of $\sqrt{n}$ (n is the dimension of model parameters) compared to [2], and our final bound is now independent of n.

Experimental results on two large-scale real-world datasets show that AD-OGP runs substantially faster than its synchronous counterpart, with nearly negligible loss in performance. We further design elaborate experiments to verify the effectiveness of the mechanisms of asymmetric gossiping and instantaneous model averaging in AD-OGP. Specifically, with either mechanism removed or weakened, the reduced algorithm will be very ineffective in some specific settings. The theoretical results of AD-OGP's regret bound on different graphs and delay levels are also well confirmed.

2 Asynchronous Decentralized Online Convex Optimization

In this section, we formally describe the AD-OCO framework. Our framework is defined over an undirected graph $G = (V, E)$ with vertex set $V = \{1, \ldots, m\}$ and edge set E. Each node $i \in V$ represents an individual learner which maintains a model $\boldsymbol{w}_i \in \mathcal{K}$ ($\mathcal{K}$ is a convex compact decision set). Each learner i can only communicate with its immediate neighbors $\mathcal{N}(i) = \{j \in V \mid (i, j) \in E\}$.

We first illustrate the whole process of AD-OCO. As the process is very complex, we divide its description into two parts: (i) *Individual view*, which specifies the action sequence of the individual learner; (ii) *System view*, which characterizes the interaction dynamics among different learners.

Individual View Each learner $i \in V$ is selected adversarially by the environment to make predictions at certain time points. We index the prediction rounds of learner i as $\tau = 1, \ldots, N_i$. At each round τ, learner i *predicts* with its most recent model $\boldsymbol{w}_i^\tau$. Then the environment reveals a convex loss function $f_i^\tau : \mathcal{K} \to \mathbb{R}$, and learner i suffers the loss $f_i^\tau(\boldsymbol{w}_i^\tau)$. When receiving the feedback, learner i begins to calculate the gradient $\nabla f_i^\tau(\boldsymbol{w}_i^\tau)$. After finishing the calculation, it uses the gradient to *update* its model weight $\boldsymbol{w}_i \leftarrow \mathcal{U}(\boldsymbol{w}_i; \nabla f_i^\tau(\boldsymbol{w}_i^\tau))$, where $\mathcal{U}$ is the local update scheme. Then it selects a subset of neighbors, and *sends* a copy of its updated model to each of them. Besides sending out its own model copies, learner i will also *receive* its neighbors' model copies at certain time points. These copies will be stored in its receiving buffer $\mathcal{B}_i$. Whenever learner i is not occupied with feedback processing or local update, and its buffer is not empty, it can *average* its model with the model copies stored in the buffer, i.e., $\boldsymbol{w}_i \leftarrow \mathcal{A}(\boldsymbol{w}_i; \mathcal{B}_i)$, where $\mathcal{A}$ is the model averaging scheme.

System View Due to the asynchronization mechanism, each learner executes its actions independently. Hence the orders of actions of all learners are intermixed. To precisely characterize the interaction dynamics among learners, we sort out the complete order of the actions of all learners by mapping them into a single time axis. To this end, we propose a novel event indexing system. We view either a prediction or a weight update of any learner as an *event*, and use a virtual counter t to index these events. Alternatively speaking, the counter t will increase by 1 each time some learner makes a prediction or completes a local update. Let $T = \sum_{i \in V} N_i$ be the total number of predictions of all learners, then the total number of local updates is also T. For each event $t \in \{1, \ldots, 2T\}$, we use $\delta_t \in \{0, 1\}$ to distinguish the event type: (i) $\delta_t = 0$ if it is a prediction event, and (ii) $\delta_t = 1$ if it is a local update event. We further use i_t to denote the learner that executes such event t, and f_t to denote the associated feedback; then there exists some $\tau \in \{1, \ldots, N_{i_t}\}$ such that $f_t = f_{i_t}^\tau$.

In particular, if event t is a local update event (i.e., $\delta_t = 1$), we denote l_t as the index of the associated prediction event [1], and $d^g(t) = t - l_t$ as the gap between these two indices. Since any prediction must occur before its corresponding local update, here we have $t \geq 2$, $l_t < t$ and $d^g(t) > 0$. Let $\text{card}(A)$ denote the cardinality of set A. Now we can introduce a quantity of f_t called the **processing delay** $d^p(t) = \text{card}\{l_t < s < t \mid \delta_s = 1\}$, which measures the number of local updates that occur between the prediction and the local update associated with f_t. Note that, our definition of processing delay is in line with those in other asynchronous frameworks for stochastic optimization [17, 3].

Recall that, after any local update event t, the learner i_t that executes this event will send out its model copy to a subset $\mathcal{S}_t$ of its neighbors $\mathcal{N}(i_t)$. We focus on the copy sent to any neighbor $j \in \mathcal{S}_t$. We assume such copy is used to average learner j's model between events r_{tj} and $(r_{tj} + 1)$ for some $r_{tj} \geq t$. For the message sent from learner i_t to learner j after event t, we introduce a quantity termed the **message delay** $d_{i_t j}^m(t) = r_{tj} - t$, which measures the number of events that occur while this copy is in transmission or learner j's receiving buffer. Note that, in the asynchronous stochastic setting [3, 23], the message delay is measured by solely counting the number of local updates. In the online setting, we also count the predictions, because whether model averaging occurs before or after a prediction will affect the gradient calculation, which further affects the corresponding weight update.

With the above event indexing system, we can now characterize the evolution of model weights $\boldsymbol{w}_i$. For each learner i, we denote $\boldsymbol{w}_i(t)$ as its most recent model before event t, and $\boldsymbol{u}_i(t)$ as its model immediately after event t. We first give the detailed characterization of local update. Specifically, for each local update event t, it is executed by learner i_t. The associated gradient was computed by learner i_t's model $\boldsymbol{w}_{i_t}(l_t)$ at prediction event l_t. So this local update on i_t can be characterized as

$$\boldsymbol{u}_{i_t}(t) = \mathcal{U}(\boldsymbol{w}_{i_t}(t); \nabla f_{l_t}(\boldsymbol{w}_{i_t}(l_t))).$$

The models of other learners do not change at i_t's local update event, so $\boldsymbol{u}_j(t) = \boldsymbol{w}_j(t)$ for $j \neq i_t$.

[1] Note that, in our indexing system, when $\delta_t = 1$, the feedback and its associated learner can be represented by the index of either the prediction event l_t or the local update event t, i.e., $f_{l_t} = f_t$ and $i_{l_t} = i_t$.

Protocol 1 Asynchronous Decentralized Online Convex Optimization (**AD-OCO**)

Input: Time horizon T, convex set $\mathcal{K}$, local update scheme $\mathcal{U}$, and model averaging scheme $\bar{\mathcal{A}}$.
Initialize: Local models $\boldsymbol{w}_j(1) \in \mathcal{K}, \forall j \in V$.
for $t = 1, \ldots, 2T$ **do**
  if $\delta_t = 0$ **then** *// for a prediction event*
    A learner $i_t \in V$ is selected by the environment adversarially to predict with $\boldsymbol{w}_{i_t}(t)$.
    Learner i_t receives a convex function f_t from environment, and suffers the loss $f_t(\boldsymbol{w}_{i_t}(t))$.
    Learner i_t starts to compute the gradient $\nabla f_t(\boldsymbol{w}_{i_t}(t))$.
  else *// for a local update event*
    Learner i_t performs local update: $\boldsymbol{u}_{i_t}(t) = \mathcal{U}(\boldsymbol{w}_{i_t}(t); \nabla f_{l_t}(\boldsymbol{w}_{i_t}(l_t))$.
    Learner i_t selects a subset $\mathcal{S}_t$ of its neighbors $\mathcal{N}(i_t)$ and sends $\boldsymbol{u}_{i_t}(t)$ to each $j \in \mathcal{S}_t$.
  end if
  for $i \in V$ **do**
    Learner i performs model averaging if it is *not* occupied: $\boldsymbol{w}_i(t+1) = \bar{\mathcal{A}}(\boldsymbol{u}_i(t); \mathcal{M}_i(t))$.
  end for
end for

We then give a full description of model averaging. Recall that, in our indexing system, model averaging operation is not regarded as an event. Hence we need to specify it using the indices of its nearby events. Concretely, for each learner i, we consider its model averaging during the interval between event t and $t+1$. If it is *not* occupied in this interval, we assume it performs s model averaging operations for some $s \geq 0$, each time using copies in $\mathcal{B}_i^1(t), \ldots, \mathcal{B}_i^s(t)$. Recall that learner i's models at the beginning and the end of such interval are $\boldsymbol{u}_i(t)$ and $\boldsymbol{w}_i(t+1)$ respectively. Hence the overall effect of model averaging on learner i between event t and $t+1$ can be expressed as

$$\boldsymbol{w}_i(t+1) = \begin{cases} \boldsymbol{u}_i(t), & s = 0; \\ \mathcal{A}(\cdots(\mathcal{A}(\mathcal{A}(\boldsymbol{u}_i(t), \mathcal{B}_i^1(t)), \mathcal{B}_i^2(t))\cdots), \mathcal{B}_i^s(t)), & s \geq 1. \end{cases}$$

Such an expression can be rewritten in a compact form. Let $\mathcal{M}_i(t) = \bigcup_{k=1}^{s} \mathcal{B}_i^k(t)$ (we set $\mathcal{M}_i(t) = \emptyset$ if $s = 0$), then we can abbreviate the above expression to $\boldsymbol{w}_i(t+1) = \bar{\mathcal{A}}(\boldsymbol{u}_t(t); \mathcal{M}_i(t))$.

Regret Definition In AD-OCO, the regret w.r.t. an arbitrary reference learner $j \in V$ is defined as

$$\mathbf{Regret}_j = \sum_{t \in \mathcal{P}_T} f_t(\boldsymbol{w}_j(t)) - \sum_{t \in \mathcal{P}_T} f_t(\boldsymbol{w}^*).$$

Here $\mathcal{P}_T = \{t \leq 2T \mid \delta_t = 0\}$ identifies all prediction events, and $\boldsymbol{w}^* \in \operatorname{argmin}_{\boldsymbol{w} \in \mathcal{K}} \sum_{t \in \mathcal{P}_T} f_t(\boldsymbol{w})$ denotes the best fixed model in hindsight. The goal of the learners is to make a sequence of predictions $\{\boldsymbol{w}_i(t)\}_{i \in V, t \in \mathcal{P}_T}$, so that the regret is sublinear in T, i.e., $\lim_{T \to \infty} \mathbf{Regret}_j / T = 0$, for any reference learner $j \in V$. Note that, our regret is in line with the regret defined in the synchronous setting [27, 31], where all the losses $\{f_t \mid t \in \mathcal{P}_T\}$ are evaluated by a fixed reference learner j.

3 Asynchronous Decentralized Online Gradient-Push

In this section, we first present the AD-OGP algorithm, then provide theoretical analysis for it.

3.1 The Algorithm

AD-OGP consists of two core modules: a local update scheme and a communication scheme. The local update scheme builds upon online gradient descent (OGD) [34]. We first present our communication scheme based on push-sum [19], then introduce two novel algorithmic designs when combining push-sum with OGD, namely weighted projection and instantaneous model averaging.

Push-Sum Communication Before presenting the details of push-sum [14], we first give some background knowledge to help understand the rationale behind it. In most decentralized algorithms, communication is based on distributed averaging [26, 9], i.e., computing the average model $\bar{\boldsymbol{x}} = \frac{1}{m} \sum_{i \in V} \boldsymbol{x}_i$ of all nodes $V = \{1, \ldots, m\}$ over a network G. In general gossip-based approaches,

at step $l \in \{1, \dots, L\}$, each learner $i \in V$ accesses the models of its neighbors and averages these models with its own model to obtain $\boldsymbol{x}_i(l+1) = \sum_{j \in V} p_{ij}(l)\boldsymbol{x}_j(l)$, where $p_{ij}(l) \geq 0$ is the averaging weight. In particular, $p_{ij}(l) > 0$ only if learner i accesses learner j's model. The model averaging of all learners has a compact matrix form $\boldsymbol{X}(l+1) = \boldsymbol{P}(l)\boldsymbol{X}(l)$, where the i-th row of $\boldsymbol{X}(l)$ represents learner i's model $\boldsymbol{x}_i(l)$, and the entry at i-th row and j-th column of $\boldsymbol{P}(l)$ is $p_{ij}(l)$.

In symmetric gossiping [9, 17], it is a common practice to design communication schemes such that the corresponding $\boldsymbol{P}(l)$ is doubly stochastic, i.e., $\boldsymbol{P}(l)\mathbf{1} = \boldsymbol{P}(l)^\top \mathbf{1} = \mathbf{1}$, where $\mathbf{1}$ is the $m \times 1$ all-one vector. Then by Markov chain theory, $\lim_{L\to\infty} \prod_{l=1}^{L} \boldsymbol{P}(l) = \frac{1}{m}\mathbf{1}\mathbf{1}^\top$, which implies that the local model of each learner i will approach $\boldsymbol{x}_i(\infty) = \bar{\boldsymbol{x}}$. The doubly stochastic property is often achieved by letting $p_{ij}(t) = p_{ji}(t)$, which requires local synchronization between the gossiping pair (i, j). In asymmetric gossiping, however, we typically have $p_{ij}(l) \neq p_{ji}(l)$. So in general, it is very hard or even impossible to design a scheme in which $\boldsymbol{P}(l)$ is doubly stochastic [2]. The usual way [20, 2] to bypass such harsh requirements is to design a communication scheme whose $\boldsymbol{P}(l)$ only satisfies the column stochasticity, i.e., $\boldsymbol{P}(l)^\top \mathbf{1} = \mathbf{1}$. Then it can be proved that $\lim_{L\to\infty} \prod_{l=1}^{L} \boldsymbol{P}(l) = \boldsymbol{\pi}\mathbf{1}^\top$ for some $\boldsymbol{\pi} \in \mathbb{R}_+^m$, which implies that each learner i will converge to $\pi_i \bar{\boldsymbol{x}}$. Push-sum strategy [19] removes the bias π_i via very light extra communication. Specifically, each learner i maintains a positive scalar y_i termed the push-sum weight, initialized as $y_i(1) = 1$. Then it applies the same averaging operation of $\boldsymbol{x}_i$ to y_i at each step l, i.e., $\boldsymbol{y}(l+1) = \boldsymbol{P}(l)\boldsymbol{y}(l)$. By the property of Markov chain, $\lim_{l\to\infty} y_i(l) = \pi_i$ for each learner i. Thus the ratio $\boldsymbol{x}_i(l)/y_i(l)$ approximately approaches $\bar{\boldsymbol{x}}$.

Given the above background knowledge, we now provide the details to implement push-sum in the asynchronous online setting. The core is to assign the averaging weights $p_{ij}(t)$ at each step t to make $\boldsymbol{P}(t)$ column stochastic. For simplicity, we first assume there is no message delay. After each local update event t, learner i_t can only send its model copy to its neighbors $\mathcal{N}(i_t)$. Hence $p_{ki_t}(t) > 0$ only if $k \in \mathcal{N}(i_t) \cup \{i_t\}$. To ensure $\sum_{k\in V} p_{ki_t}(t) = 1$, a natural choice is to assign the *same* weight $p_{ki_t}(t) = \gamma_{i_t}$ for all $k \in \mathcal{N}(i_t) \cup \{i_t\}$, where $\gamma_{i_t} = 1/(|\mathcal{N}(i_t)| + 1)$. Our implementation satisfies this condition: After each local update t, learner i_t first multiplies its model $\boldsymbol{x}_{i_t}$ and push-sum weight y_{i_t} by γ_{i_t}, which become $\boldsymbol{x}'_{i_t} = \gamma_{i_t}\boldsymbol{x}_{i_t}$ and $y'_{i_t} = \gamma_{i_t} y_{i_t}$. Then it sends a copy of $\boldsymbol{x}'_{i_t}$ and y'_{i_t} to each neighbor $j \in \mathcal{N}(i_t)$. In the corresponding model averaging, learner j adds $\boldsymbol{x}'_{i_t}$ and y'_{i_t} to its own parameters, i.e., $\boldsymbol{x}_j \leftarrow \boldsymbol{x}_j + \boldsymbol{x}'_{i_t} = \boldsymbol{x}_j + \gamma_{i_t}\boldsymbol{x}_{i_t}$ and $y_j \leftarrow y_j + y'_{i_t} = y_j + \gamma_{i_t} y_{i_t}$. The model averaging operation of learner i using buffer $\mathcal{B}_i^s$ can be summarized as

$$\mathcal{A}((\boldsymbol{x}_i, y_i); \mathcal{B}_i^s) = (\boldsymbol{x}_i \ + \sum_{(\gamma_j \boldsymbol{x}_j, \gamma_j y_j) \in \mathcal{B}_i^s} \gamma_j \boldsymbol{x}_j, \ y_i \ + \sum_{(\gamma_j \boldsymbol{x}_j, \gamma_j y_j) \in \mathcal{B}_i^s} \gamma_j y_j).$$

Note that, when there are message delays, our scheme can still induce column stochastic $\boldsymbol{P}(t)$ on the augmented graph [10] at each step $t \in \{1, \dots, 2T\}$ (see detailed proofs in the appendix).

Weighted Projected Online Gradient Descent Recall that, in push-sum, each learner $i \in V$ has two types of parameters, namely model $\boldsymbol{x}_i$ and push-sum weight y_i; and it always predicts with $\boldsymbol{w}_i = \boldsymbol{x}_i/y_i$. Learner i is initialized by $\boldsymbol{x}_i \leftarrow \boldsymbol{w}_0 \in \mathcal{K}$ and $y_i \leftarrow 1$. In OGD's update with gradient $\boldsymbol{g}$, learner i first performs a gradient step on its model $\boldsymbol{x}_i$, i.e., $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \eta\boldsymbol{g}$, where η is the learning rate. However, its next prediction $\boldsymbol{x}_i/y_i$ may lie outside of the decision set $\mathcal{K}$, which is not feasible.

To ensure feasibility, OGD executes a projection operation $\Pi_{\mathcal{K}}$ on the model $\boldsymbol{x}_i$ after the gradient step, i.e., $\Pi_{\mathcal{K}}(\boldsymbol{x}_i) = \operatorname{argmin}_{\boldsymbol{x}\in\mathcal{K}} \|\boldsymbol{x} - \boldsymbol{x}_i\|$ ($\|\cdot\|$ means the L2 norm). However, if we directly apply the standard projection, we may still have $\boldsymbol{w}_i = \Pi_{\mathcal{K}}(\boldsymbol{x}_i)/y_i \notin \mathcal{K}$ after projection, because y_i can be less than 1. Hence we need a special design of projection operation. Notice that, when $y_i > 0$, $\boldsymbol{x}_i/y_i \in \mathcal{K}$ is equivalent to $\boldsymbol{x}_i \in y_i\mathcal{K} = \{y_i\boldsymbol{w} \mid \boldsymbol{w} \in \mathcal{K}\}$. Since $y_i\mathcal{K}$ is also convex, we can project $\boldsymbol{x}_i$ onto $y_i\mathcal{K}$ instead of $\mathcal{K}$. Then we have $\Pi_{y_i\mathcal{K}}(\boldsymbol{x}_i) = \operatorname{argmin}_{\boldsymbol{x}\in y_i\mathcal{K}} \|\boldsymbol{x} - \boldsymbol{x}_i\| = y_i \operatorname{argmin}_{\boldsymbol{w}\in\mathcal{K}} \|\boldsymbol{w} - \boldsymbol{x}_i/y_i\| = y_i\Pi_{\mathcal{K}}(\boldsymbol{x}_i/y_i)$, which can be viewed as the standard projection $\Pi_{\mathcal{K}}(\boldsymbol{x}_i)$ distorted by weight y_i. The above local update procedure can be summarized as $\boldsymbol{x}_i \leftarrow y_i\Pi_{\mathcal{K}}((\boldsymbol{x}_i - \eta\boldsymbol{g})/y_i)$ and $y_i \leftarrow y_i$.

Recall that, in our implementation of push-sum, we multiply learner i's parameters $\boldsymbol{x}_i$ and y_i by a factor of γ_i *immediately* after its local update. This multiplication operation can be combined with the above local update form, which gives a compact expression

$$\mathcal{U}((\boldsymbol{x}_i, y_i); \boldsymbol{g}) = (\gamma_i y_i \Pi_{\mathcal{K}}((\boldsymbol{x}_i - \eta\boldsymbol{g})/y_i), \gamma_i y_i).$$

Instantaneous Model Averaging When we tailor asynchronization to the online setting, another design problem arises from the online process, namely *when* to perform model averaging $\mathcal{A}$ for each learner. In the online setting, a learner is unoccupied only when it completes a local update and waits to be selected by the environment to make the next prediction. Hence the learner can only perform model averaging during this period. On the other side, the learner may receive messages at any time. Each message will be stored in the receiving buffer until the learner performs the next model averaging. The waiting time of this message also contributes to its message delay. To reduce message delays to the most extent, we allow each learner to perform model averaging as soon as possible. (i) If the learner receives any message during gradient calculation or weight update, it will use this message for model averaging once it completes the current local update. (ii) If the learner receives any message during its unoccupied period, it will perform model averaging immediately.

We remark that, instantaneous model averaging is a novel design specific to asynchronous online learning. In fact, in stochastic optimization [3, 23], when to perform model averaging does not even constitute a problem. In that setting, after each local update, the time for the next "prediction" is actively decided by the learner. In other words, the learner can begin to process the next sample immediately after it completes the current local update. Hence the learner is always occupied, and the only time available for model averaging is when it just finishes a local update. This significantly simplifies the action orders of all learners, which makes the asynchronous algorithm much easier to design and analyze in the stochastic setting than that in the online setting.

3.2 Theoretical Analysis

We first consider the special case of $m = 1$. For the only learner, its push-sum weight is constantly $y_1(t) = 1$, and its model evolves as $\boldsymbol{x}_1(t+1) = \Pi_{\mathcal{K}}(\boldsymbol{x}_1(t) - \eta \nabla f_t(\boldsymbol{x}_1(t)))$, which exactly reduces to the standard OGD [34]. Therefore, in the following, we focus on $m \geq 2$. We introduce two mild assumptions, which are commonly used in the analysis of other asynchronous settings [3, 23].

Assumption 1. *(a) The message delays are commonly bounded by some integer $D^{msg} \geq 0$.*
(b) Each learner performs local update at least once every Γ_d steps for some integer $\Gamma_d > 0$.

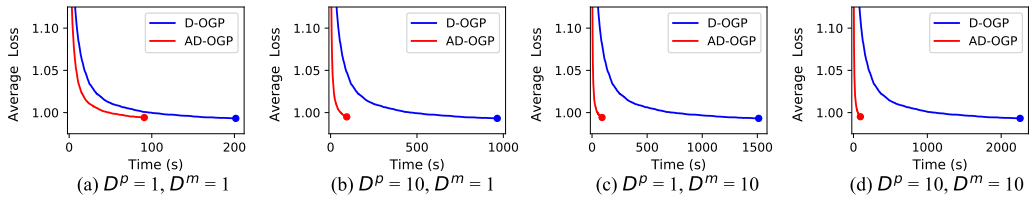
Before presenting the regret bound, we give some notations and definitions. We use $\mathcal{Q}_T = \{t \leq 2T \mid \delta_t = 1\} = \{1, \ldots, 2T\} - \mathcal{P}_T$ to denote the indices of all local update events. Among the events in $\mathcal{Q}_T$, we further use $\mathcal{Q}_{s,t} = \mathcal{Q}_T \cap (s, t)$ to identify those that occur between events s and t. Now for any $t \in \mathcal{Q}_T$, the processing delay $d^p(t)$ of f_t is exactly $|\mathcal{Q}_{l_t,t}|$. Also denote the gradients $\boldsymbol{g}_t = \nabla f_{l_t}(\boldsymbol{x}_{i_t}(l_t)/y_{i_t}(l_t)), \hat{\boldsymbol{g}}_t = \nabla f_{l_t}(\boldsymbol{x}_j(l_t)/y_j(l_t))$, their L2 norms $g_t = \|\boldsymbol{g}_t\|, \hat{g}_t = \|\hat{\boldsymbol{g}}_t\|$, the total processing delay of all feedbacks as $D^{proc} = \sum_{t \in \mathcal{Q}_T} d^p(t)$, and the diameter of graph G as $\mathcal{D}$.

Theorem 1. *For any reference learner $j \in V$, AD-OGP attains the following regret bound*

$$\begin{aligned}\mathbf{Regret}_j \leq \frac{m}{2\eta}\|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2 + \frac{2\eta}{m}\sum_{t \in \mathcal{Q}_T}\sum_{s \in \mathcal{Q}_{l_t,t+1}} g_t g_s + \eta \sum_{t \in \mathcal{Q}_T} \frac{g_t^2}{y_{i_t}(t)} \\ + 2\eta \sum_{t \in \mathcal{Q}_T}\sum_{s \in \mathcal{Q}_{1,l_t}} \lambda^{\lfloor \frac{l_t - s}{2B} \rfloor}(g_t + 2\hat{g}_t)\frac{g_s}{y_{i_s}(t)} + 2\eta \sum_{t \in \mathcal{Q}_T}\sum_{s \in \mathcal{Q}_{1,t+1}} \lambda^{\lfloor \frac{t-s}{2B} \rfloor} g_t \frac{g_s}{y_{i_s}(t)},\end{aligned}$$

where $\lambda = 1 - m\alpha^4, \alpha = (\frac{1}{m})^B, B = (\mathcal{D}+1)(D^{msg} + \Gamma_d)$.

Proof sketch Our proof consists of three steps. *First*, we extend graph augmentation technique [10] to the online setting to handle message delays. Specifically, we add D^{msg} virtual nodes for each node $i \in V$, and derive a column stochastic $\boldsymbol{P}(t)$ at each step $t \in \{1, \ldots, 2T\}$ on the augmented graph. *Second*, we sort out the complex effects of prediction, weight update, and communication of all learners via our proposed event indexing system. Then we derive a general regret bound in terms of transition matrix $\boldsymbol{P}(t)$ at each step $t \leq 2T$ and gradients $\boldsymbol{g}_t, \hat{\boldsymbol{g}}_t$ at each local update event $t \in \mathcal{Q}_T$. *Third*, we analyze the asymptotic properties of $\boldsymbol{P}(t)$ induced by our push-sum strategy. This part of analysis is enlightened by prior works [20, 23], but we also make an extra contribution to the convergence analysis of push-sum. Specifically, when analyzing the diffusion of each local update over the network, we tackle all of its n dimensions simultaneously by utilizing the fact that the rank



(a) $D^p = 1, D^m = 1$ (b) $D^p = 10, D^m = 1$ (c) $D^p = 1, D^m = 10$ (d) $D^p = 10, D^m = 10$

Figure 1: Illustrating the benefit of asynchronization. The four plots compare AD-OGP and D-OGP on 64-node Watts-Strogatz graphs on *pokerhand* under varying processing/message delays D^p/D^m.

of the updating matrix $\mathbf{\Delta}(t) = \boldsymbol{g}_t \boldsymbol{e}_{i_t}^\top$ is at most one. In this way, we eliminate the factor $\sqrt{n}$ in the convergence rate in prior analysis [2] (in their coefficient C). Our bound is now independent of n.

The above bound can be further simplified under the bounded gradient norm assumption [27]:

Corollary 1. *Suppose $g_t \leq G, \hat{g}_t \leq G, \forall t \in \mathcal{Q}_T$ for some $G < \infty$. Then AD-OGP attains the regret*

$$\mathbf{Regret}_j \leq \frac{m}{2\eta}\|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2 + \frac{2\eta}{m}G^2(T + D^{proc}) + \frac{2\eta}{m}(\frac{8+\alpha^4}{\alpha^5})BG^2T,$$

for any reference learner $j \in V$, where $\alpha = (1/m)^B, B = (\mathcal{D}+1)(D^{msg} + \Gamma_d)$.

Remark (a) Suppose the diameter of decision set $\mathcal{K}$ is bounded by F. Set $\eta = (mF/2G)(D^{proc} + T + CBT)^{-1/2}$ where $C = (8+\alpha^4)/\alpha^5$, and then bound will attain $O((CBT + T + D^{proc})^{1/2})$, which will be larger for more learners (larger m), heavier processing delays (larger D^{proc}), heavier message delays (larger D^{msg}), and "less connected" graphs (larger diameter $\mathcal{D}$).
(b) When processing delays $d^p(t)$ are upper bounded, i.e., $D^{proc} = O(T)$, the above derived bound will reduce to $O((CBT)^{1/2})$, which is sublinear $O(\sqrt{T})$ w.r.t. T.
(c) The second term in the bound captures the effect of processing delays $\sum_{t\in\mathcal{Q}_T} d^p(t)$. This term is in line with the delay effect term in the asynchronous centralized online setting [18].
(d) The third term in the bound captures the effect of decentralization. Note that, the factor α appears as a lower bound of the positive entries in any B-hop transition matrix $\boldsymbol{P}(t+B-1)\cdots\boldsymbol{P}(t+1)\boldsymbol{P}(t), t \in \{1,\ldots,2T-B\}$, which is commonly seen in the analysis of push-sum [1, 20, 2, 3].
(e) When there are no delays, i.e., $D^{proc} = D^{msg} = 0$, the bound reduces to that of its synchronous counterpart [1] (for a fair comparison, we remove the assumption of strongly convex losses in [1]).

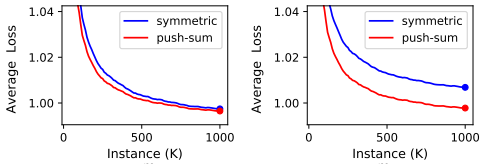
4 Experiments

In this section, we evaluate AD-OGP on two large-scale real-world online classification tasks. We first compare AD-OGP with its synchronous counterpart to show the advantage of asynchronization, then design elaborate experiments to verify the effectiveness of asynchronous communication and instantaneous model averaging in our algorithm design, and finally verify our theoretical regret bound.

4.1 Experimental Setup

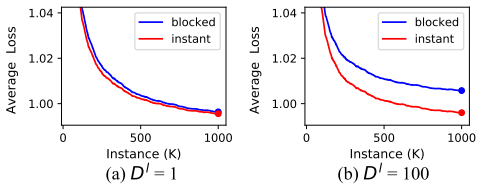
We select two large-scale real-world datasets. (i) The *higgs* dataset is a benchmark dataset in high-energy physics [4] for binary classification, which consists of 11 million instances with 28 features. (ii) The *poker-hand* dataset is a commonly used dataset in automatic rule generation [6, 7] for 10-class classification, which has 1 million instances with 25 features. For binary classification, we use the logistic loss; for multi-class classification, we adopt the multivariate logistic loss [11] (see details for their definitions in the appendix). We use *average loss* [31] as the performance metric: we randomly select a reference learner $j \in V$ and measure $\sum_{t\in\mathcal{P}_T} f_t(\boldsymbol{w}_j(t))/T$, where $\boldsymbol{w}_j(t) = \boldsymbol{x}_j(t)/y_j(t)$. We adopt the common L2-norm balls as decision sets and set the diameters $F = 100$. The learning rate η is set as what the corresponding theory suggests (see more details in appendix).

The computation time of each feedback and the transmission time of each message are both simulated by the approach in [25], which provides strong reproducibility of empirical results. Specifically,



(a) $D^m = 1$ (b) $D^m = 100$

Figure 2: Illustrating the effectiveness of asymmetric gossiping. The two plots compare push-sum and symmetric gossiping [17] on a 64-node Watts-Strogatz graph on *pokerhand*. We increase the transmission time of any message sent from/to reference learner j by D^m times.



(a) $D^l = 1$ (b) $D^l = 100$

Figure 3: Illustrating the effectiveness of instantaneous model averaging. We compare AD-OGP and the algorithm with blocked model averaging [3] on a 64-node Watts-Strogatz graph on *pokerhand*. We inject a time lag δD^l to reference learner j after each of its local update.

the computation time on learner i is sampled from an exponential distribution $\exp(\mu_i)$ with an expectation of $\mu_i = |\bar{\mu}| + 2$, where $\bar{\mu}$ follows a normal distribution $N(0, 1)$. The transmission time of each message is independently sampled from $\exp(0.6)$. Both are in the unit of millisecond. Note that, we also examine other delay patterns such as uniform distributions and observe similar results.

4.2 The Benefit of Asynchronization

We compare AD-OGP with synchronous decentralized online gradient-push (D-OGP) [1], which synchronizes local weight update and push-sum communication of all learners at each iteration. The experiments are conducted on three representative types of graphs with varying connectivities [9]: complete graph of high connectivity, Watts-Strogatz graph as a kind of random graph of medium connectivity, and ring graph of low connectivity (see appendix for detailed descriptions).

To investigate the effect of processing delays and message delays on the algorithms' performance, we adopt a technique commonly used in asynchronous experiments [17, 23]. (i) For the reference learner j, we enlarge its computation time by D^p times throughout the learning process, where D^p is a parameter that controls the processing delay level. (ii) We randomly select a neighbor k of the reference learner j and increase the transmission time of any message on edge (j, k) by D^m times, where D^m is a parameter that controls the message delay level. Similar to [17], we set D^p (or D^m) $= 1$ to represent a weak delay level, and set D^p (or D^m) $= 10$ to represent a strong delay level.

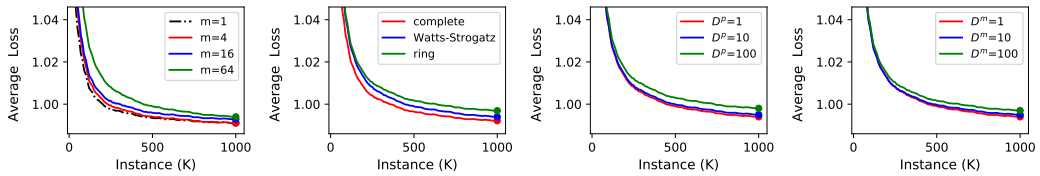
Figure 1 plots the runtime performance of both algorithms on a 64-node Watts-Strogatz graph under varying levels of processing and message delays on *pokerhand*. The results on *higgs* as well as other graphs are qualitatively similar and deferred to the appendix due to space limit. From the results, we have two main observations. (i) AD-OGP runs substantially faster than D-OGP, in accord well with our intuition that asynchronization eschews the straggler problem and saves considerable time. Moreover, the gap in total running time considerably enlarges when delays become stronger. (ii) Along with its benefit in efficiency, AD-OGP yields comparable performance to D-OGP. In summary, asynchronization realizes high efficiency while hardly sacrificing the algorithm's performance.

4.3 The Effectiveness of Asymmetric Gossiping and Instantaneous Model Averaging

We verify the effectiveness of the two designs in AD-OGP by showing that, with either mechanism removed or weakened, the reduced algorithm will become very ineffective in some specific settings.

4.3.1 Effectiveness of Asymmetric Gossiping

We consider a scenario where communication via some learner j is slow. In symmetric gossiping [17], each time learner j sends a model copy to any of its neighbor $k \in \mathcal{N}(j)$, it must wait until receiving the response message copy sent back from k, before it can proceed to the next round; so its weight update and communication frequency is slowed down. In push-sum, the local update and model averaging frequency of this learner is not much affected. To verify this point empirically, we enlarge the transmission time of any message sent from/to the reference learner j by D^m times. We compare



(a) network size (b) network topology (c) processing delays (d) message delays

Figure 4: Verifying our regret bound. The four plots show AD-OGP's performance on *poker-hand*. Experiments are conducted on 64-node Watts-Strogatz graph with delay levels $D^p = D^m = 1$, except for: (a) Varying network sizes m. ($m = 1$ reduces to OGD [34].) (b) Different network topologies. (c) Varying levels of processing delays D^p. (d) Varying levels of message delays D^m.

AD-OGP and the counterpart algorithm using symmetric gossiping (with [17]'s implementation) on 64-node Watts-Strogatz graph under varying D^m. Results in Figure 2 show that symmetric gossiping becomes ineffective as message delays increase, while asymmetric gossiping is much more robust.

4.3.2 Effectiveness of Instantaneous Model Averaging

We consider a setting where some learner j is seldom selected by the environment to make predictions. In this case, after learner j completes a local update, it may wait for a long time until its next prediction. During this period, learner j stays unoccupied and may continuously receive model copies from its neighbors. In AD-OGP, these copies are used for model averaging immediately after their arrivals. In contrast, if we directly apply other asynchronous stochastic algorithms [3, 23] to our online setting, these copies will be stored in learner j's receiving buffer until it makes the next local update. This essentially magnifies the message delays and further increases the regret. To verify this point, we inject a time lag of $\delta \cdot D^l$ to the reference learner j each time it completes a local update and prepares to make the next prediction. Here δ is independently sampled from $\exp(0.6)$, and D^l is a parameter. We deploy AD-OGP and the counterpart algorithm using [3]'s implementation on a 64-node Watts-Strogatz graph under varying D^l. The results in Figure 3 show that instantaneous model averaging is very effective when prediction and local update of some learner are infrequent.

4.4 Verification of Theoretical Bound

We verify our derived regret bound by examining the effect of the various factors on the performance of AD-OGP. Specifically, we run AD-OGP under varying network sizes, different network topologies, and varying levels of processing delays and message delays. Figure 4 presents the results on *poker-hand*. The results show that AD-OGP will suffer from higher loss with more learners, less connected network (which implies a larger diameter), heavier processing delays, and heavier message delays, which is consistent with our theoretical conclusions in Corollary 1.

5 Conclusions and Limitations

In this paper, we conduct the first systematic study of asynchronous decentralized online learning. We first formulate the framework of Asynchronous Decentralized Online Convex Optimization. Then we devise the Asynchronous Decentralized Online Gradient-Push algorithm, which is fully asynchronous and has two special designs when tailoring asynchronization to online learning, namely weighted projection and instantaneous model averaging. Theoretically, we provide the first paradigm of regret analysis for asynchronous decentralized online learning. We finally conduct extensive experiments to demonstrate the benefit of asynchronization and verify theoretical results.

Limitations As the first step of studying asynchronous learning in the decentralized online setting, our framework formulation and theoretical analysis are conducted in the convex setting. Although such limitation does not affect our practice usage or empirical results (a non-convex case is also studied in the appendix), we would like to establish a non-convex analysis in future works.

References

[1] Mohammad Akbari, Bahman Gharesifard, and Tamás Linder. Distributed online convex optimization on time-varying directed graphs. *IEEE Transactions on Control of Network Systems*, 4(3):417–428, 2015.

[2] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Mike Rabbat. Stochastic gradient push for distributed deep learning. In *International Conference on Machine Learning*, pages 344–353. PMLR, 2019.

[3] Mahmoud Assran and Michael Rabbat. Asynchronous gradient-push. *IEEE Transactions on Automatic Control*, 2020.

[4] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

[5] Yixin Bao, Yanghua Peng, Chuan Wu, and Zongpeng Li. Online job scheduling in distributed machine learning clusters. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 495–503. IEEE, 2018.

[6] Robert Cattral, Franz Oppacher, and Dwight Deugo. Evolutionary data mining with automatic rule generalization. *Recent Advances in Computers, Computing and Communications*, 1(1):296–300, 2002.

[7] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[8] John Duchi, Michael I Jordan, and Brendan McMahan. Estimation, optimization, and parallelism when data is sparse. In *Advances in Neural Information Processing Systems*, pages 2832–2840, 2013.

[9] John C Duchi, Alekh Agarwal, and Martin J Wainwright. Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Transactions on Automatic control*, 57(3):592–606, 2011.

[10] Christoforos N Hadjicostis and Themistoklis Charalambous. Average consensus in the presence of delays in directed graph topologies. *IEEE Transactions on Automatic Control*, 59(3):763–768, 2013.

[11] Elad Hazan and Haipeng Luo. Variance-reduced and projection-free stochastic optimization. In *International Conference on Machine Learning*, pages 1263–1271, 2016.

[12] Hadrien Hendrikx, Francis Bach, and Laurent Massoulié. Accelerated decentralized optimization with local updates for smooth and strongly convex objectives. In *AISTATS 2019-22nd International Conference on Artificial Intelligence and Statistics*, 2019.

[13] Zhanhong Jiang, Aditya Balu, Chinmay Hegde, and Soumik Sarkar. Collaborative deep learning in fixed topology networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5906–5916, 2017.

[14] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 482–491. IEEE, 2003.

[15] Alec Koppel, Santiago Paternain, Cédric Richard, and Alejandro Ribeiro. Decentralized online learning with kernels. *IEEE Transactions on Signal Processing*, 66(12):3240–3255, 2018.

[16] Jinlong Lei, Peng Yi, Yiguang Hong, Jie Chen, and Guodong Shi. Online convex optimization over erdos-renyi random networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[17] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pages 3049–3058, 2018.

[18] Brendan McMahan and Matthew Streeter. Delay-tolerant algorithms for asynchronous distributed online learning. In *Advances in Neural Information Processing Systems*, pages 2915–2923, 2014.

[19] Angelia Nedić and Alex Olshevsky. Distributed optimization over time-varying directed graphs. *IEEE Transactions on Automatic Control*, 60(3):601–615, 2014.

[20] Angelia Nedić and Alex Olshevsky. Stochastic gradient-push for strongly convex functions on time-varying directed graphs. *IEEE Transactions on Automatic Control*, 61(12):3936–3947, 2016.

[21] Santiago Paternain, Soomin Lee, Michael M Zavlanos, and Alejandro Ribeiro. Distributed constrained online learning. *arXiv preprint arXiv:1903.06310*, 2019.

[22] S Shahrampour and A Jadbabaie. Distributed online optimization in dynamic environments using mirror descent. *IEEE Transactions on Automatic Control*, 63(3):714–725, 2018.

[23] Artin Spiridonoff, Alex Olshevsky, and Ioannis Ch Paschalidis. Robust asynchronous stochastic gradient-push: Asymptotically optimal and network-independent performance for strongly convex functions. *Journal of Machine Learning Research*, 21(58):1–47, 2020.

[24] Yuanyu Wan, Wei-Wei Tu, and Lijun Zhang. Projection-free distributed online convex optimization with $o(\sqrt{t})$ communication complexity. In *International Conference on Machine Learning*, pages 9818–9828. PMLR, 2020.

[25] Tianyu Wu, Kun Yuan, Qing Ling, Wotao Yin, and Ali H Sayed. Decentralized consensus optimization with asynchrony and delays. *IEEE Transactions on Signal and Information Processing over Networks*, 4(2):293–307, 2017.

[26] Lin Xiao and Stephen Boyd. Fast linear iterations for distributed averaging. *Systems & Control Letters*, 53(1):65–78, 2004.

[27] Feng Yan, Shreyas Sundaram, SVN Vishwanathan, and Yuan Qi. Distributed autonomous online learning: Regrets and intrinsic privacy-preserving properties. *IEEE Transactions on Knowledge and Data Engineering*, 25(11):2483–2493, 2013.

[28] Xinlei Yi, Xiuxian Li, Lihua Xie, and Karl H Johansson. Distributed online convex optimization with time-varying coupled inequality constraints. *IEEE Transactions on Signal Processing*, 68:731–746, 2020.

[29] Chi Zhang, Peilin Zhao, Shuji Hao, Yeng Chai Soh, Bu Sung Lee, Chunyan Miao, and Steven CH Hoi. Distributed multi-task classification: A decentralized online learning approach. *Machine Learning*, 107(4):727–747, 2018.

[30] Ruiliang Zhang and James Kwok. Asynchronous distributed admm for consensus optimization. In *International Conference on Machine Learning*, pages 1701–1709, 2014.

[31] Wenpeng Zhang, Peilin Zhao, Wenwu Zhu, Steven CH Hoi, and Tong Zhang. Projection-free distributed online learning in networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4054–4062. JMLR. org, 2017.

[32] Yawei Zhao, Chen Yu, Peilin Zhao, and Ji Liu. Decentralized online learning: Take benefits from others' data without sharing your own to track global trend. *arXiv preprint arXiv:1901.10593*, 2019.

[33] Zhengyuan Zhou, Panayotis Mertikopoulos, Nicholas Bambos, Peter W Glynn, and Claire Tomlin. Countering feedback delays in multi-agent learning. In *Advances in Neural Information Processing Systems*, pages 6171–6181, 2017.

[34] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 928–936, 2003.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No] We study a mathematical problem. The societal impact largely depends on specific real-world application scenarios.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Stated in Assumption 1.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code and data source are provided in supplementary materials.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] (i) The online learning setup does not require data splits. (ii) How the hyperparameters were chosen is clarified in Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A] (i) Our algorithm is deterministic; (ii) In the online learning setup, the sequence of instances is fixed.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cited the creators of datasets.
   (b) Did you mention the license of the assets? [Yes] We mention it in Appendix.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We only use open source datasets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We only use open source datasets.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]