# RWDDA: Distributed Dual Averaging Made Practical

**Cun Mu***, **Asim Kadav**†, **Erik Kruus**†, **Donald Goldfarb***, **Martin Renqiang Min**†
NEC Laboratories America†
Columbia University*

## Abstract

The increased data collection and processing abilities of embedded devices allows learning across these devices for novel applications such as traffic management, power generation etc. However, real world distributed machine learning across multiple devices suffers from intermittent device and link failures, and data bias skews that make existing distributed optimization methods less useful.

We propose a failure resilient distributed learning framework called RWDDA, that provides provable convergence in the presence of failures and non-i.i.d. data. RWDDA improves upon the existing distributed dual averaging (DDA) method, making it robust to changes in network topology and amenable to asynchronous implementations. RWDDA also uses several implementation optimizations that provide our dual-order methods about $200X$ speedup making them as fast as primal methods. Our theoretical analysis shows the algorithm has $O(1/\sqrt{t})$ convergence for non-smooth convex problems. We apply our techniques to distributed SVM and find that our system outperforms competing methods, especially in the presence of communication link failures.

## 1 Introduction

Machine learning algorithms are commonly being applied across sensors, mobile and personal devices and even geographically distributed data centers. These devices generate and share the data and the computational costs of training a model. However, distributed learning in real world scenarios suffers from two issues. First, the various nodes in a real-world setting may suffer from intermittent network or node failures. For example, geographically separated data centers may suffer from communication delays or dropped packets. Second, the nodes in the distributed system such as the physical sensors may collect data points that are not randomly distributed across the nodes resulting in non-independent and identically distributed (non-i.i.d.) data across the nodes. Data-centers too, often collect non-random data, with each data center receiving data that is biased towards the geography where it is located. Often due to scale, privacy, or lack of a central coordinating resource, randomizing data may not always be possible. As a result, distributed training across these nodes in the presence of biased data at individual machines, based on simple techniques such as averaging of parameters may not work.

The most commonly applied paradigm for distributed machine learning is the parameter server approach [9, 10, 20]. The parameter server approach has reasonable computational costs and resets the parameters for all workers ensuring consensus based convergence. However, for many practical embedded sensors there may not be a central reliable server. Furthermore, if the server fails, the recovery requires using complex protocols [17]. Hence, an alternative approach is to perform distributed consensus optimization methods by using all-reduce style methods [18]. This approach is commonly used with MPI, Hadoop and other map-reduce style frameworks [1]. This model has no separate master and avoids any centralized bottlenecks [8]. Furthermore, a failed node may simply

*Work done as a NEC Labs intern. Now at Columbia University.

be removed from the training process. An all-reduce approach is preferred when data is generated at separate physical sources (such as geographically distributed data-centers or sensors).

The all-reduce approach does not require maintaining a set of reliable master servers using complex fault tolerance protocols such as Paxos that are required for master-slave systems. Furthermore, the all-reduce approach only requires communicating gradients for convex problems and reduces high-dimensional model communication costs.

A disadvantage of the all-reduce technique is that it requires communicating with all peers. Hence, for N machines, this results in O($N^2$) communication complexity. Because of these prohibitive communication costs, these systems may communicate infrequently that results in poor convergence [41]. To address these issues, second order techniques such as dual-averaging have been proposed that perform sparse communication but still guarantee convergence [13, 14, 33]. However, these methods are not resilient to failures, and cannot handle dynamic change in network topologies. Furthermore, the dual-order methods are expensive to use, and simple averaging is preferred to aggregate the model updates.

To address above problems, we build a system called RWDDA, that makes distributed dual averaging practical. RWDDA is capable of handling dynamic communication networks, since each node in RWDDA only needs to know about their neighborhood in the network based on received model updates, rather than global properties of the entire network. As a result, RWDDA can handle real-world intermittent failure scenarios. RWDDA also provides several practical performance optimizations that make it 200X faster than existing dual methods. To the best of our knowledge, our work is the first such implementation to show that a dual method can work as fast as a primal method. Furthermore, these optimizations can also be applied to other dual-order methods that are considered impractical due to their slow speeds.

## 2 Background

### 2.1 Problem Statement

In mathematical terms, the optimization problem is defined on a connected undirected network and solved by $n$ agents (computers) collectively,

$$\min_{\boldsymbol{x}\in\mathcal{X}\subseteq\mathbb{R}^d} \quad \bar{f}(\boldsymbol{x}) := \sum_{i=1}^{n} f_i(\boldsymbol{x}). \tag{2.1}$$

The feasible set $\mathcal{X}$ is a closed and convex set in $\mathbb{R}^d$ and is commonly known by all agents, whereas $f_i : \mathcal{X} \in \mathbb{R}$ is a convex function privately known by the agent $i$. Throughout the paper, we also assume that $f_i$ is $L$-Lipschitz continuous over $\mathcal{X}$ with respect to the Euclidean norm $\|\cdot\|$. The network $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, with the node set $\mathcal{N} = [n] := \{1, 2, \cdots, n\}$ and the edge set $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$, specifies the topological structure on how the information can be spread among agents through local agent interactions over time. In specific, each agent $i$ can only send and retrieve information from its neighbors $\mathcal{N}(i) := \{j \mid (j, i) \in \mathcal{E}\}$ and itself.

### 2.2 Related Work

In this paper, we propose to solve this problem in the framework of *Decentralized Consensus Optimization (DCO)*, where all the nodes (agents), with their own utility functions, are connected through a network. The networked system goal is to optimize the sum of all the utility functions, only through local computations and local information exchange with neighbors as specified by the communication graph of all nodes. Such peer-to-peer framework, with applications ranging from large scale machine learning [33, 22, 21, 39] to wireless sensor networks [25, 12], tends to be scalable, simple to implement and robust to intermittent network failures.

Given the importance of DCO, many methods have been proposed recently [23]. One of the techniques that gained recent popularity is a class of distributed algorithms that combine the consensus protocols developed from the control field [28] and the gradient-type methods from the optimization area [27]. Here, a consensus protocol refers to a mechanism for information diffusion, where each agent independently spread their information via locally weighted averaging of their incoming data. The gradient-based methods are particularly suitable, since they, in general, have a

small per-iteration cost and are robust to various sources of stochastic errors. In contrast with approaches based on bi-directional communication, e.g. Randomized Gossiping [7] and ADMM-type distributed methods [36], the above mentioned technique employs only one-directional communication, (where agents only send information and then proceed with their local computations without expecting a response from others,) and thus manage to avoid the deadlock problem resulting from the bi-direction communication in practical implementation.

Generally speaking, based on the style of the consensus step, these methods can be classified as model averaging methods [25, 29, 30, 38, 15, 19, 32], which average parameters, and dual averaging methods [3, 33, 35], which average (sub)gradients. Arguably, the dual averaging approach is preferable as it is more scalable with the size of the network than the primal one [3]. However, the dual method may suffer from significant performance overheads. In specific, the dual computation and distributed consensus lead to high CPU costs when computing the dual variable every iteration, which makes it impractical when applied to large datasets as compared to primal model averaging methods.

Moreover, the successes of the above mentioned model averaging methods and dual averaging methods heavily rely on the communication network to be static, which may not be realistic due to node/edge failures. One exception is these methods [34, 33, 24, 40] using the push-sum protocol, aka weighted gossip or sum-weight algorithms [16, 4, 14]. In theory, push-sum type methods are robust to the change in network topology under the synchronous scenario. However, as pointed in [33], due to the scaling issue, these methods are often numerical instable in practical asynchronous implementation. Our numerical experiment in Section 4 also conforms to this observation.

# 3 RWDDA for resilient consensus optimization

## 3.1 RWDDA system design

In this section, we describe RWDDA design. RWDDA is designed to support multiple replicas across different nodes training on different data. After training an example (or a batch of examples), these nodes send model information to one-another and learn using RWDDA's dual averaging algorithm. Hence, with RWDDA each machine computes a new model based on the data examples, and performs a *push* operation to its peers as decided by the network topology. It them performs a *reduce* operation over the received model updates and proceeds to train over more data.

In RWDDA, each machine maintains a per-sender queue to receive the model information that can be written to by the peers using shared memory protocols. This design of using dedicated per-sender queues allows the senders to write to its receivers without any coordination. Hence, when using one-sided protocols, such as RDMA or GPUDirect, this operation can be performed asynchronously without any synchronization from even the receiving node's hardware. This architecture ensures one-sided communication and scales better as the number of nodes increase. Furthermore, this style of communication is optimal for dual-averaging methods that send updates using one-sided undirected graphs and do not need to perform a consensus operation to ensure convergence.

We now describe the parallel learning algorithm used by RWDDA based on the theory of random walk over connected graphs.

## 3.2 RWDDA algorithm

RWDDA algorithm is shown step-by-step in Algorithm 2. In this algorithm, each node $i$ keeps a local estimate $\boldsymbol{x}_i$ and a dual variable $\boldsymbol{z}_i$ maintaining an accumulated subgradient. At iteration $t$, to update $\boldsymbol{z}_i$, each node needs to collect the $\boldsymbol{z}$-values of its neighbors, forms a convex combination with equal weight of the received information and adds its most recent local subgradient scaled by $|\mathcal{N}(i)| + 1$. After that, the dual variables $\boldsymbol{z}_i$ is projected to the primal space to obtain $\boldsymbol{x}_i$.

To implement the RWDDA method, each node only needs to know its neighborhood information, which makes the algorithm robust to any changes in the network topology, which may occur frequently from node or edge failures. As possibly inferred from the name, RWDDA method robustifies the distributed dual averaging (DDA) method [3], based on the theory of random walk over undirected graph [31]. However, the original DDA method heavily relies on the fact that the consensus matrix is doubly stochastic , which requires regular node communication graphs. This

**Algorithm 1** RWDDA Method

**Input:** a predetermined non-negative non-increasing sequence $\{\alpha(t)\}$.
**Initialization:** $\boldsymbol{x}_i(0) = \boldsymbol{z}_i(0) = \mathbf{0}$, for all $i \in [n]$.
**for** $t = 0, 1, 2, \ldots,$ **do**

1. Subgradient calculation:

$$\boldsymbol{g}_i(t) \in \partial f_i(\boldsymbol{x}_i(t)), \text{ for each agent } i. \tag{3.1}$$

2. Dual updates:

$$\boldsymbol{z}_i(t+1) = \frac{\sum_{j\in\mathcal{N}(i)\cup\{i\}} \boldsymbol{z}_j(t) + \boldsymbol{g}_i(t)}{|\mathcal{N}(i)|+1}, \tag{3.2}$$
$$\text{for each agent } i.$$

3. Primal updates:

$$\begin{aligned}\boldsymbol{x}_i(t+1) &= \mathcal{P}_{\mathcal{X}}[-\alpha(t)\boldsymbol{z}_i(t+1)] \\ &:= \arg\min_{\boldsymbol{x}\in\mathcal{X}} \ \|\boldsymbol{x} + \alpha(t)\boldsymbol{z}_i(t+1)\|^2,\end{aligned} \tag{3.3}$$
$$\text{for each agent } i.$$

**end for**

requirement restricts the DDA method from being fully distributed, especially when facing changes in network topology. Specifically, whenever a link or node failure happens, the DDA method has to stop the algorithm and reconfigure its consensus matrix using a central master node, which makes it impractical in real settings such as in the case of peer-to-peer sensors. To overcome this difficulty, RWDDA method self-adjusts the consensus matrix at a feasible local level, by using a row stochastic matrix plus a gradient scaling trick. RWDDA requires only that the communication matrix $P$ is row stochastic, which allows non-regular communication graphs. We establish the theory for RWDDA method by borrowing ideas from the random walk theory from stochastic process.

We briefly describe the intuition behind the correctness of our DDA. We provide the convergence proof for Algorithm 2 in supplementary materials.

For notational convenience, we will define the matrix $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ with $P_{ij}$ being $\frac{1}{|\mathcal{N}(i)+1|}$ for $j \in \mathcal{N}(i)\cup\{i\}$ and 0 otherwise. Clearly, $\boldsymbol{P}$ is a row stochastic matrix, i.e. the sum of every row of $\boldsymbol{P}$ equals 1. We will also define the vector $\boldsymbol{\pi} \in \mathbb{R}^d$ with the $i$-th entry $\pi_i$ being $\frac{|\mathcal{N}(i)|+1}{\beta}$, where $\beta := 2|\mathcal{V}| + |\mathcal{E}|$. It can easily verified that $\boldsymbol{\pi}$ is a probability vector, i.e. $\pi_i > 0$ and $\sum_{i\in[n]} \pi_i = 1$. With these notations, we are able to express (6.2) in a terser way. Imagine $\mathcal{X} \subseteq \mathbb{R}$, so $x_i(t)$, $z_i(t)$ and $g_i(t)$ are now all scalars. Then we can rewrite the update (6.2) as

$$\begin{aligned}\boldsymbol{z}(t+1) &= \boldsymbol{P}\boldsymbol{z}(t) + \frac{1}{\beta}\mathrm{diag}\,(\boldsymbol{\pi})^{-1}\boldsymbol{g}(t) \\ &= \frac{1}{\beta}\sum_{s=0}^{t} \boldsymbol{P}^s \mathrm{diag}\,(\boldsymbol{\pi})^{-1}\boldsymbol{g}(t-s),\end{aligned} \tag{3.4}$$

with $\boldsymbol{z}(t) = (z_1(t), z_2(t), \cdots, z_n(t))^\top$ and $\boldsymbol{g}(t) = (g_1(t), g_2(t), \cdots, g_n(t))^\top$. As we need each node to play the same role in the system, from (3.4), it is quite reasonable to require $\boldsymbol{P}^\infty \mathrm{diag}\,(\boldsymbol{\pi})^{-1} = \mathbb{1}_{n\times n}$, where $\boldsymbol{P}^\infty := \lim_{t\to\infty} \boldsymbol{P}^t$ and $\mathbb{1}_{n\times n}$ is the $n$ by $n$ matrix with all entries as one. Indeed, we can verify this requirement by the close connection between $\boldsymbol{P}$ and $\boldsymbol{\pi}$, as revealed in the following lemma, which can be regarded as a direct consequence of results for random walk under undirected graph [31]. This also justifies the appearance of random walk in the name of our algorithm.

### 3.3 RWDDA system improvements

For our RWDDA implementation, on every node $k$, we loop over the local training examples. For every iteration $t$, we choose an example $i$, and calculate the local gradient $\boldsymbol{g}_i$ and update the current model $\boldsymbol{x}_k^t$. In distributed optimization across multiple nodes, we perform a push operation of the computed gradients and perform a reduce operation on the received gradients. In the reduce step, we sum any incoming gradient contributions (dual vectors) as $\boldsymbol{z}_k^{t'} = \sum_{j \in \mathcal{I}_k} \boldsymbol{z}_j$ and incorporate gradient $\boldsymbol{g}_i$ into dual $\boldsymbol{z}_k$ as $\boldsymbol{z}_k^{t+1} = \frac{\boldsymbol{z}_k^{t'} + \boldsymbol{g}_i}{|\mathcal{I}_k|+1}$. After processing a batch of examples on every machine (about 500-5000), we push dual gradient $z_k^{t+1}$ via out-edges $\mathcal{O}_k$. We also choose learning rate $\eta_k^t$ and apply the dual gradient $\boldsymbol{z}_k^{t+1}$ as $\boldsymbol{x}_k^{t+1} = -\eta_k^t \cdot \boldsymbol{z}_i$. Finally, each node also maintains and updates the running average or the consensus model as $\hat{\mathbf{x}}_k^{t+1} = \sum_{i=1}^{t+1} \boldsymbol{x}_k^i/(t+1) = \frac{t}{t+1}\hat{\mathbf{x}}_k^t + \frac{1}{t+1}\boldsymbol{x}_k^{t+1}$.

To improve performance, we perform the following three optimizations.

1. First, instead of calculating the full gradient on every iteration, we only compute the sparse gradient and separately correct the regularizer [6].
2. Second, instead of sending $z$ (or $w$ for model averaging) after every update step to all other nodes, we send it infrequently to reduce communication costs. Each node locally processes examples (usually 500-5000), and then communicates $z$. We adjust the learning rate parameter $\eta$ to account for the batched communication.
3. Finally, we maintain a running sum average over the dual, and only compute this sum only during reduce (incoming $z$ parameters). Furthermore, in our asynchronous implementation if there are no incoming dual variables ($z$), we skip updating the average. We find that the above optimizations give us significant speedups (over $200X$) allowing the dual space algorithms that we implement, to operate as fast as primal space algorithms.

## 4 Experiments

RWDDA provides distributed machine learning over shared memory for Leon Bottou's SVM-SGD [5]. We implement RWDDA simple model averaging and PS-DDA over SVM SGD. We implement model averaging such that each machine calculates the partial gradient and sends it to other machines via a *push operation*. Each machine averages the received gradients in a *reduce* step and updates its model weight vector($w$) locally. In our implementation, RWDDA and model averaging communicate variables in a one-sided fashion without requiring an acknowledgment and we use the one-sided primitives over RDMA to perform this low latency communication.

We now evaluate the RWDDA algorithm for training SVM using the RCV1 and the webspam dataset [2]. We evaluate RWDDA according to the following criteria:

1. *Performance:* How does RWDDA compare with existing primal and dual methods? We evaluate performance for the case when data is not randomly distributed across the machines (non-i.i.d case) with dense and sparse networks.
2. *Fault tolerance:* How does RWDDA behave with non-i.i.d. data and in the presence of link failures?

We perform all experiments on an eight machine research cluster connected via an infiniBand backplane. We run multiple processes, across these four machines, and we refer to each process as a rank (from the HPC terminology). We run multiple ranks on each machine, especially for models with less than 1M parameters, where a single model replica is unable to saturate the network and CPU. Each machine has an Intel Xeon 8-core, 2.2 GHz Ivy-Bridge processor with support for SSE 4.2/AVX instructions, and 64 GB DDR3 DRAM. Each machine is connected via a Mellanox Connect-V3 56 Gbps infiniBand cards. Our 56 Gbps infiniBand network architecture provides a peak throughput of slightly over 40 Gbps after accounting for the bit-encoding overhead for reliable transmission. All machines share storage using a 10 TB NFS partition that we use for loading input data. Each process loads a portion of data depending on the number of processes. For all our experiments, we partition the input data and assign positive or negative subsets to each node. Hence, we perform all our training with a sampling bias over non-i.i.d data unless mentioned otherwise. All

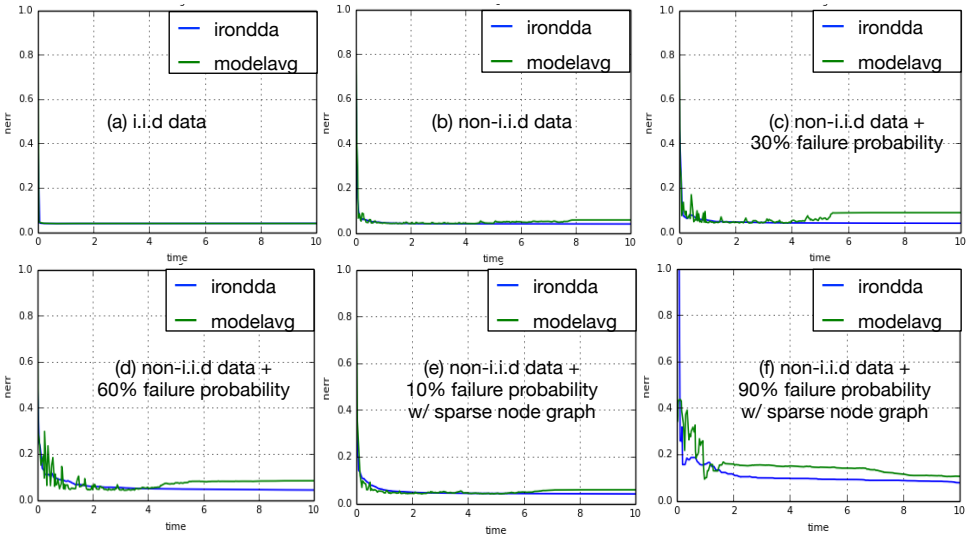


Figure 1: **This figure shows the convergence of RWDDA with model averaging for 6 parallel ranks (processes across machines) and all ranks exchange parameters with one another for the RCV1 dataset. Each rank communicates $z$ or $w$ after processing a local epoch (slightly over 3300 examples). Figure (a) shows performance over i.i.d data where RWDDA and model averaging compare favorably. Figure (b) shows this performance for non-i.i.d data. Figure (c) and (d) show convergence comparisons for 30% and 60% probability of packet losses. We find that RWDDA converges in both cases. Figures (e) and (f) illustrate performance comparisons for sparse node graph where each machine only exchanges parameters with $N/2$ machines.**

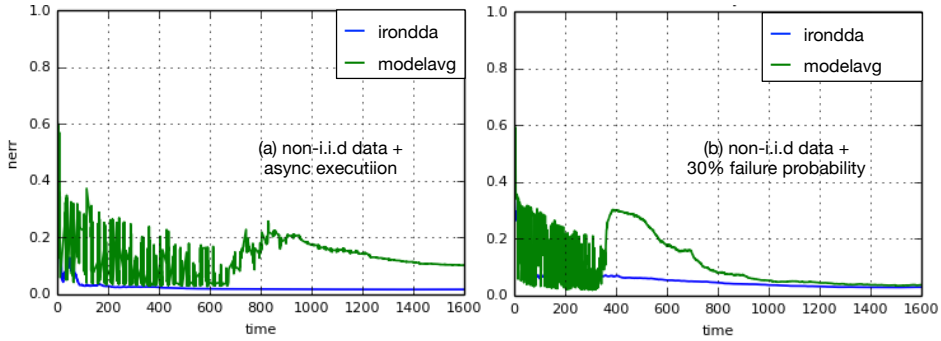


Figure 2: **This figure shows the convergence of RWDDA as compared to model averaging for 16 parallel ranks (processes across machines) and all ranks exchange parameters with one another for the webspam dataset. Each rank communicates $z$ or $w$ after processing a local epoch (slightly over 5000 examples). Figure (a) shows asynchronous performance over non-i.i.d data where RWDDA and model averaging. Figure (b) shows this performance for non-i.i.d data and 30% probability of packet losses. We find that RWDDA converges correctly in both cases.**

reported times do not account the initial one-time cost for the loading the data-sets in memory. All times are reported in seconds.

We compare RWDDA and model averaging with the applicable optimizations described in the previous section. Model averaging is computationally efficient because of its simple update step. We also implement failure resiliency in both the algorithms by appropriately detecting the number of nodes sending parameters and correctly computing a scaling factor. We run all our experiments over six ranks. Each rank represents a process that may span multiple machines and each rank trains over a subset of data. For our experiments, the six ranks span across three machines.

We compare the average training error over all ranks w.r.t. wall clock time in Figures 1 (a) and (b). In this section, we compare the performance of RWDDA and model averaging without failures. We choose a densely connected network graph where each machine synchronizes parameters with all other machines. Figure (a) shows the convergence for i.i.d. data where both model averaging and

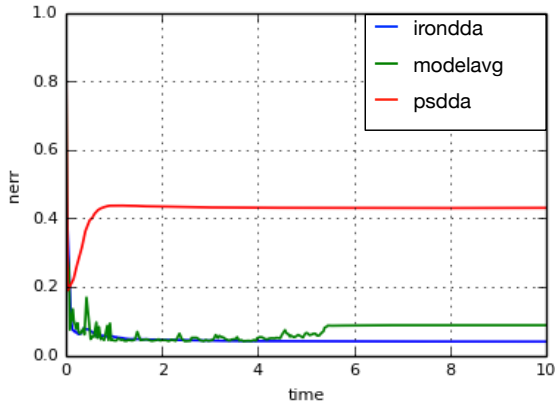


Figure 3: **This figure compares model-averaging, RWDDA and PS-DDA for 30% probability of packet loss. We demonstrate results for average convergence across ranks and we find that PS-DDA suffers from numerical instability in the scalar that results in incorrect convergence.**

RWDDA converge correctly. For non-i.i.d. data, we find that RWDDA converges faster in time, and achieves a stable accuracy better than model averaging. Hence, we find that with our optimizations, our dual-order method, RWDDA, performs as good as primal order model averaging. From the optimizations described in previous section, we are able to obtain more than $200X$ speedup from the original RWDDA implementation. Furthermore, for the non-i.i.d. dataset, RWDDA converges correctly unlike model averaging.

### 4.1 Fault Tolerance

We now compare RWDDA performance in the presence of intermittent link failures. Each outgoing packet may fail with a specific user-defined probability. The failures are asymmetric i.e. nodes with positive examples are less likely to fail than those with negative examples. We repeat our experiments for different overall failure probability goals. For our fault tolerance experiments, we remove the barrier before the update step, since some packets may never arrive due to failures and a barrier will lead to infinite wait. Hence, we perform our fault tolerance experiments by running the algorithms asynchronously.

Figures 1 (c) and (d) show RWDDA and model convergence with 30% and 60% packet loss probability where all machines communicate with one-another forming a dense communication graph of nodes. We find that RWDDA is more robust to link failures and offers correct convergence which model averaging is unable to achieve. To account for fewer incoming $z$ parameters due to the asynchrony, we appropriately re-scale the gradient or the averaging fraction by counting the number of incoming $z$ or $g$ parameters.

We now provide performance comparisons with undirected sparse communication graphs i.e. where all nodes may not communicate with one another. Instead of communicating with all other machines (or processes), each machine only communicates with $N/2$ other machines such that the network graph of all machines is connected and the graph is undirected, where $N$ is the total number of nodes. We compare RWDDA and model-averaging over a sparse communication graph in Figure 1 (e) and (f) with 10% and 90% packet loss probability. We find that model averaging does not converge correctly in Figure 1 (f) while RWDDA achieves correctly.

Figures 2 (a) and (b) shows this comparison with the webspam dataset consisting of 16.6M parameters, running over 16 processes. We find that for large datasets too, RWDDA is robust against asynchrony or failures in the presence of non-i.i.d. datasets.

**Comparisons with PS-DDA** We implement Push-Sum DDA [33] in our framework and apply the same optimizations as RWDDA to improve its performance. Figure 3 compares model averaging, RWDDA and PS-DDA for failures with 30% probability of packet loss for a specific rank, with non-i.i.d. data. PS-DDA requires sending an additional scaling component. Additionally, in the asynchronous case or in presence of failures, PS-DDA suffers from numerical instability [33]. In asynchronous mode, since different nodes operate at different speeds, the scalar may become very

small due to repeated re-scaling. To prevent this from happening, in our implementation, we reset the scalar to its initial value (1.0) if it becomes too large or too small. As a result, of the numerical instability we find that PS-DDA is unable to converge in the presence of packet losses and non-i.i.d. data. However, we find that PS-DDA performs comparably with RWDDA in absence of failures (not shown in figure).

To summarize, from our evaluation we find that RWDDA has good convergence properties and our implementation of RWDDA is robust and efficient.

## 5 Conclusion

Distributed learning over a large number of distributed sensors or geographically separated data centers suffers from sampling biases and communication link failures. Existing dual averaging approaches are slow, and may not converge correctly in the presence of link-failures, which are fairly common in real world deployments. This happens because these algorithms such as DDA make the assumption that the communication/transition matrix $P$ is doubly stochastic, which requires regular node communication graphs.

We present RWDDA a distributed learning algorithm that is robust to failures. RWDDA requires only that the communication matrix $P$ is row stochastic, which allows non-regular communication graphs. This graph structure then allows an easy weighting scheme to maintain convergence to the correct fixed points. Our analysis shows the algorithm has $O(1/\sqrt{t})$ convergence for non-smooth convex problems. Our experiments show that RWDDA converges as fast as primal averaging algorithms and provides smooth convergence.

## References


[1] Vowpal Wabbit. `http://hunch.net/ vw/`.

[2] PASCAL Large Scale Learning Challenge. `http://largescale.ml.tu-berlin.de`, 2009.

[3] A. Agarwal and J. C. Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, 2011.

[4] F. Bénézit, V. Blonde, P. Thiran, J. Tsitsiklis, and M. Vetterli. Weighted gossip: Distributed averaging using non-doubly stochastic matrices. In *Information theory proceedings (isit), 2010 ieee international symposium on*. IEEE, 2010.

[5] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Springer COMPSTAT*, 2010.

[6] L. Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[7] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE/ACM Transactions on Networking (TON)*, 14(SI):2508–2530, 2006.

[8] W. Chen, Z. Wang, and J. Zhou. Large-scale l-bfgs using mapreduce. In *Advances in Neural Information Processing Systems*, 2014.

[9] W. Dai, J. Wei, X. Zheng, J. K. Kim, S. Lee, J. Yin, Q. Ho, and E. P. Xing. Petuum: A framework for iterative-convergent distributed ml. *arXiv preprint arXiv:1312.7651*, 2013.

[10] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, et al. Large scale distributed deep networks. In *NIPS*, 2012.

[11] P. Diaconis and D. Stroock. Geometric bounds for eigenvalues of markov chains. *The Annals of Applied Probability*, pages 36–61, 1991.

[12] A. G. Dimakis, S. Kar, J. Moura, M. G. Rabbat, and A. Scaglione. Gossip algorithms for distributed signal processing. *Proceedings of the IEEE*, 98(11):1847–1864, 2010.

[13] J. C. Duchi, A. Agarwal, and M. J. Wainwright. Dual averaging for distributed optimization: convergence analysis and network scaling. *Automatic Control, IEEE Transactions on*, 57(3):592–606, 2012.

[14] F. Iutzeler, P. Ciblat, and W. Hachem. Analysis of sum-weight-like algorithms for averaging in wireless sensor networks. *Signal Processing, IEEE Transactions on*, 61(11):2802–2814, 2013.

[15] D. Jakovetic, J. Xavier, and J. M. Moura. Fast distributed gradient methods. *Automatic Control, IEEE Transactions on*, 59(5):1131–1146, 2014.

[16] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*. IEEE, 2003.

[17] L. Lamport. Generalized consensus and paxos. Technical report, Technical Report MSR-TR-2005-33, Microsoft Research, 2005.

[18] J. Langford, A. Smola, and M. Zinkevich. Slow learners are fast. *arXiv preprint arXiv:0911.0491*, 2009.

[19] H. Li, A. Kadav, E. Kruus, and C. Ungureanu. Malt: distributed data-parallelism for existing ml applications. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015.

[20] M. Li, D. Andersen, A. Smola, J. Park, A. Ahmed, V. Josifovski, J. Long, E. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *USENIX OSDI*, 2014.

[21] Q. Ling, Z. Wen, and W. Yin. Decentralized jointly sparse optimization by reweighted minimization. *Signal Processing, IEEE Transactions on*, 61(5):1165–1170, 2013.

[22] Q. Ling, Y. Xu, W. Yin, and Z. Wen. Decentralized low-rank matrix completion. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012.

[23] A. Nedić. Distributed optimization. In *Encyclopedia of Systems and Control*, pages 1–12. Springer London, 2014.

[24] A. Nedic and A. Olshevsky. Distributed optimization over time-varying directed graphs. *Automatic Control, IEEE Transactions on*, 60(3):601–615, 2015.

[25] A. Nedić and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *Automatic Control, IEEE Transactions on*, 54(1):48–61, 2009.

[26] Y. Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical programming*, 120(1):221–259, 2009.

[27] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[28] A. Olshevsky and J. N. Tsitsiklis. Convergence speed in distributed consensus and averaging. *SIAM Journal on Control and Optimization*, 48(1):33–55, 2009.

[29] S. S. Ram, A. Nedić, and V. V. Veeravalli. Distributed stochastic subgradient projection algorithms for convex optimization. *Journal of optimization theory and applications*, 147(3):516–545, 2010.

[30] S. S. Ram, A. Nedić, and V. V. Venugopal. A new class of distributed optimization algorithms: Application to regression of distributed data. *Optimization Methods and Software*, 27(1):71–88, 2012.

[31] S. Ross. *Stochastic processes*, volume 2. John Wiley & Sons New York, 1996.

[32] W. Shi, Q. Ling, G. Wu, and W. Yin. Extra: An exact first-order algorithm for decentralized consensus optimization. *SIAM Journal on Optimization*, 25(2):944–966, 2015.

[33] K. Tsianos, S. Lawlor, and M. G. Rabbat. Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*. IEEE, 2012.

[34] K. Tsianos, S. Lawlor, and M. G. Rabbat. Push-sum distributed dual averaging for convex optimization. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*. IEEE, 2012.

[35] K. Tsianos and M. G. Rabbat. Distributed dual averaging for convex optimization under communication delays. In *American Control Conference (ACC), 2012*. IEEE, 2012.

[36] E. Wei and A. Ozdaglar. On the $O(1/k)$ convergence of asynchronous distributed alternating direction method of multipliers. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*. IEEE, 2013.

[37] L. Xiao. Dual averaging method for regularized stochastic learning and online optimization. In *Advances in Neural Information Processing Systems*, 2009.

[38] K. Yuan, Q. Ling, and W. Yin. On the convergence of decentralized gradient descent. *arXiv preprint arXiv:1310.7063*, 2013.

[39] K. Yuan, Q. Ling, W. Yin, and A. Ribeiro. A linearized bregman algorithm for decentralized basis pursuit. In *Signal Processing Conference (EUSIPCO), 2013 Proceedings of the 21st European*. IEEE, 2013.

[40] J. Zeng and W. Yin. Extrapush for convex smooth decentralized optimization over directed networks. *arXiv preprint arXiv:1511.02942*, 2015.

[41] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In *NIPS*, 2010.

# 6 Convergence Analysis

**Lemma 1.** $\boldsymbol{\pi}^\top \boldsymbol{P} = \boldsymbol{\pi}^\top$ *and* $\boldsymbol{P}^\infty := \lim_{t\to\infty} \boldsymbol{P}^t = \mathbf{1}\cdot\boldsymbol{\pi}^\top$.

*Proof.* Consider a discrete-time Markov chain with state space as $\mathcal{V}$ and transition matrix specified by $\boldsymbol{P}$. It can be easily seen that this Markov chain is irreducible and aperiodic. Therefore, there exists a unique stationary distribution $\boldsymbol{d}$ satisfying $\boldsymbol{d} \geq 0, \mathbf{1}^\top \boldsymbol{d} = 1, \boldsymbol{d}^\top \boldsymbol{P} = \boldsymbol{d}^\top$ and $\boldsymbol{P}^\infty = \mathbf{1} \cdot \boldsymbol{d}^\top$. Since the probability vector $\boldsymbol{\pi}$ satisfies the so-called detailed balance equation, i.e. $\pi_i P_{ij} = \pi_j P_{ji}$, $\boldsymbol{\pi}$ is the stationary distribution, i.e. $\boldsymbol{d} = \boldsymbol{\pi}$. □

### 6.1 RWDDA algorithm

**Algorithm 2** RWDDA Method

**Input:** a predetermined non-negative non-increasing sequence $\{\alpha(t)\}$.
**Initialization:** $\boldsymbol{x}_i(0) = \boldsymbol{z}_i(0) = \mathbf{0}$, for all $i \in [n]$.
**for** $t = 0, 1, 2, \ldots,$ **do**

1. Subgradient calculation:

$$\boldsymbol{g}_i(t) \in \partial f_i(\boldsymbol{x}_i(t)), \text{ for each agent } i. \tag{6.1}$$

2. Dual updates:

$$\boldsymbol{z}_i(t+1) = \frac{\sum_{j \in \mathcal{N}(i) \cup \{i\}} \boldsymbol{z}_j(t) + \boldsymbol{g}_i(t)}{|\mathcal{N}(i)| + 1}, \quad \text{for each agent } i. \tag{6.2}$$

3. Primal updates:

$$\begin{aligned} \boldsymbol{x}_i(t+1) &= \mathcal{P}_{\mathcal{X}}[-\alpha(t)\boldsymbol{z}_i(t+1)] \\ &:= \arg\min_{\boldsymbol{x} \in \mathcal{X}} \ \|\boldsymbol{x} + \alpha(t)\boldsymbol{z}_i(t+1)\|^2, \\ &\qquad \text{for each agent } i. \end{aligned} \tag{6.3}$$

**end for**

In this section, we will provide an $O(1/\sqrt{t})$-convergence result for Algorithm 2 when $\alpha(t)$ is properly chosen as $O(1/\sqrt{t})$.

We first present two useful lemmas. The first one is standard in convex analysis. and the second one is from [3], which modifies slightly the result in [26].

**Lemma 2.** *For any $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$, $\|\mathcal{P}_{\mathcal{X}}[\boldsymbol{u}] - \mathcal{P}_{\mathcal{X}}[\boldsymbol{v}]\| \leq \|\boldsymbol{u} - \boldsymbol{v}\|$.*

**Lemma 3.** *Let $\{\boldsymbol{h}(t)\}_{t=1}^\infty \subset \mathbb{R}^d$ be an arbitrary sequence of vectors and $\{\alpha(t)\}_{t=1}^\infty$ be a positive and non-increasing sequence. Consider the sequences $\{\bar{\boldsymbol{z}}(t)\}_{t=0}^\infty$ with $\bar{\boldsymbol{z}}(0) = \mathbf{0}$ and $\{\boldsymbol{y}(t)\}_{t=1}^\infty$ constructed as follows:*

$$\begin{aligned} &\boldsymbol{y}(t+1) = \mathcal{P}_{\mathcal{X}}[-\alpha(t)\bar{\boldsymbol{z}}(t)], \\ &\bar{\boldsymbol{z}}(t+1) = \bar{\boldsymbol{z}}(t) + \boldsymbol{h}(t), \qquad t = 0, 1, 2, \ldots. \end{aligned}$$

*Then for any $\boldsymbol{x}^\star \in \mathcal{X}$, we have*

$$\begin{aligned} &\sum_{t=1}^{T} \langle \boldsymbol{h}(t), \boldsymbol{y}(t) - \boldsymbol{x}^\star \rangle \\ &\qquad \leq \frac{1}{2} \sum_{t=1}^{T} \alpha(t-1) \|\boldsymbol{h}(t)\|^2 + \frac{1}{2\alpha(T)} \|\boldsymbol{x}^\star\|. \end{aligned}$$

Now, we can proceed to our proof for the RWDDA.

**Lemma 4.** *Consider the sequences $\{\boldsymbol{x}_i(t)\}$ and $\{\boldsymbol{z}_i(t)\}$ generated by random-walk distributed dual averaging (Algorithm 2). Then for any $\boldsymbol{x}^\star \in \mathcal{X}$ and for each node $i \in [n]$, we have*

$$\begin{aligned}
&\bar{f}\left(\widehat{\boldsymbol{x}}_i\left(T\right)\right) - \bar{f}(\boldsymbol{x}^\star) \\
&\leq \frac{L^2 n}{2\beta T}\sum_{t=1}^{T}\alpha(t-1) + \frac{\beta}{2nT\alpha(T)}\left\|x^\star\right\|^2 \\
&+ \frac{L}{T}\sum_{t=1}^{T}\alpha(t)\left(\frac{2}{n}\sum_{j=1}^{n}\left\|\bar{\boldsymbol{z}}(t)-\boldsymbol{z}_j(t)\right\| + \left\|\bar{\boldsymbol{z}}(t)-\boldsymbol{z}_i(t)\right\|\right),
\end{aligned} \tag{6.4}$$

*where $\widehat{\boldsymbol{x}}_i(T) = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{x}_i(t)$ and $\bar{\boldsymbol{z}}(t) = \sum_{i=1}^{n}\pi_i\boldsymbol{z}_i(t)$.*

*Proof.* For simplicity, we assume $\mathcal{X} \subseteq \mathbb{R}$, so $x_i(t)$, $z_i(t)$ and $g_i(t)$ are now all scalars. Denote $\boldsymbol{z}(t) = (z_1(t), \ldots, z_n(t))^\top \in \mathbb{R}^d$, and $\boldsymbol{g}(t) = (g_1(t), \ldots, g_n(t))^\top \in \mathbb{R}^d$.

Based on the dynamics (6.2), we can write

$$\boldsymbol{z}(t+1) = \boldsymbol{P}\boldsymbol{z}(t) + \frac{1}{\beta}\mathrm{diag}\left(\boldsymbol{\pi}\right)^{-1}\boldsymbol{g}(t). \tag{6.5}$$

Then one has the $\boldsymbol{\pi}$-weighted average

$$\begin{aligned}
\bar{z}(t+1) = \boldsymbol{\pi}^\top\boldsymbol{z}(t+1) &= \boldsymbol{\pi}^\top\boldsymbol{P}\boldsymbol{z}(t) + \frac{1}{\beta}\boldsymbol{\pi}^\top\mathrm{diag}\left(\boldsymbol{\pi}\right)^{-1}\boldsymbol{g}(t) \\
&= \boldsymbol{\pi}^\top\boldsymbol{z}(t) + \frac{1}{\beta}\mathbf{1}^\top\boldsymbol{g}(t) = \bar{z}(t) + \frac{1}{\beta}\mathbf{1}^\top\boldsymbol{g}(t),
\end{aligned}$$

where we have used the fact that $\boldsymbol{\pi}^\top\boldsymbol{P} = \boldsymbol{\pi}^\top$ in Lemma 1.

Now define

$$y(t+1) = \mathcal{P}_{\mathcal{X}}[-\alpha(t)\bar{z}(t)], \qquad t = 0, 1, 2, \ldots, \tag{6.6}$$

and we start to bound our target $f(\widehat{x}_i(T)) - f(x^\star)$,

$$\begin{aligned}
&f(\widehat{x}_i(T)) - f(x^\star) \\
&\leq \frac{1}{T}\sum_{t=1}^{T}\Big(f(x_i(t)) - f(x^\star)\Big) \\
&\leq \frac{1}{T}\sum_{t=1}^{T}\Big(f(y(t)) - f(x^\star)\Big) + \frac{1}{T}\sum_{t=1}^{T}\Big(f(x_i(t)) - f(y(t))\Big) \\
&\leq \frac{1}{nT}\sum_{t=1}^{T}\sum_{i=1}^{n}\Big(f_i(y(t)) - f_i(x^\star)\Big) + \frac{L}{T}\sum_{t=1}^{T}\left\|x_i(t) - y(t)\right\|,
\end{aligned} \tag{6.7}$$

where the first inequality is due to convexity, and last line holds as $f$ is $L$-Lipschitz.

Now let us fucus on the term $f_i(y(t)) - f_i(x^\star)$,

$$\begin{aligned}
&f_i(y(t)) - f_i(x^\star) \\
&= \Big(f_i(y(t)) - f_i(x_i(t))\Big) + \Big(f_i(x_i(t)) - f_i(x^\star)\Big) \\
&\leq L\left\|y(t) - x_i(t)\right\| + \langle g_i(t), x_i(t) - x^*\rangle \\
&= L\left\|y(t) - x_i(t)\right\| + \langle g_i(t), x_i(t) - y(t)\rangle + \langle g_i(t), y(t) - x^\star\rangle \\
&\leq L\left\|y(t) - x_i(t)\right\| + L\left\|y(t) - x_i(t)\right\| + \langle g_i(t), y(t) - x^\star\rangle \\
&\leq 2L\left\|y(t) - x_i(t)\right\| + \langle g_i(t), y(t) - x^\star\rangle,
\end{aligned} \tag{6.8}$$

where the second inequality holds as $f_i$ is $L$-Lipschitz and $g_i(t) \in \partial f_i(x_i(t))$, and the second last line is due to $\|g_i\| \leq L$.

Substituting (6.8) into (6.7), we obtain

$$f(\widehat{x}_i(T)) - f(x^\star) \leq \frac{1}{nT}\sum_{t=1}^{T}\sum_{i=1}^{n}\langle g_i(t), y(t) - x^\star\rangle + \frac{L}{T}\sum_{t=1}^{T}\left(\frac{2}{n}\sum_{i=1}^{n}\|y(t) - x_i(t)\| + \|y(t) - x_i(t)\|\right). \tag{6.9}$$

Next, we will look at the two terms in (6.9) respectively. For the first term,

$$\begin{aligned}
&\frac{1}{nT}\sum_{t=1}^{T}\sum_{i=1}^{n}\langle g_i(t), y(t) - x^\star\rangle \\
&= \frac{\beta}{nT}\sum_{t=1}^{T}\left\langle \frac{\mathbf{1}^\top \boldsymbol{g}(t)}{\beta}, y(t) - x^\star\right\rangle \\
&\leq \frac{\beta}{2nT}\sum_{t=1}^{T}\alpha(t-1)\left\|\frac{\mathbf{1}^\top \boldsymbol{g}(t)}{\beta}\right\|^2 + \frac{\beta}{2nT\alpha(T)}\|x^\star\|^2 \\
&\leq \frac{L^2 n}{2\beta T}\sum_{t=1}^{T}\alpha(t-1) + \frac{\beta}{2nT\alpha(T)}\|x^\star\|^2,
\end{aligned} \tag{6.10}$$

where the second line results from Lemma 3 (with $\mathbf{1}^\top \boldsymbol{g}(t)/\beta$ playing the role of $h(t)$ in Lemma 3).

Regarding the second term in (6.9), note that

$$\begin{aligned}
&\|y(t) - x_i(t)\| = \|\mathcal{P}_{\mathcal{X}}[-\alpha(t)\bar{z}(t)] - \mathcal{P}_{\mathcal{X}}[-\alpha(t)\bar{z}_i(t)]\| \\
&\leq \|-\alpha(t)(\bar{z}(t) - z_i(t))\| \leq \alpha(t)\|\bar{z}(t) - z_i(t)\|.
\end{aligned} \tag{6.11}$$

by Lemma 2.

Finally, substituting (6.10) and (6.11) into (6.9) yields our desired (6.4). □

Lemma (4) provides a nice characterization of the deviation from the optimal value over all nodes. The first two terms in (6.4) are common optimization error terms pertaining to subgradient algorithms. The third term reflects the nature of distributed optimization, where each node has its own estimate of the average gradient that deviates from each other. Next, we will show an upper bound for the deviation term $\|\bar{\boldsymbol{z}}(t) - \boldsymbol{z}_i(t)\|$.

**Lemma 5.** *Consider the sequences $\{\boldsymbol{x}_i(t)\}$ and $\{\boldsymbol{z}_i(t)\}$ generated by random-walk distributed dual averaging (Algorithm 2). Define $\bar{\boldsymbol{z}}(t) = \boldsymbol{\pi}^\top \boldsymbol{z}(t)$. Then we have,*

$$\|\bar{\boldsymbol{z}}(t) - \boldsymbol{z}_i(t)\| \leq \frac{L}{\beta\pi_{\min}}\sqrt{\frac{1-\pi_i}{\pi_i}}\frac{1}{1-\sigma_2(\boldsymbol{P})}, \tag{6.12}$$

*where $\pi_{\min} = \min\{\pi_i\}$ and $\sigma_2(\cdot)$ denotes the second largest singular value.*

*Proof.* For simplicity, we assume $\mathcal{X} \subseteq \mathbb{R}$, so $x_i(t)$ and $z_i(t)$ are now scalars. Denote $\boldsymbol{z}(t) = (z_1(t), z_2(t), \cdots, z_n(t))^\top$ and $\boldsymbol{g}(t) = (g_1(t), g_2(t), \cdots, g_n(t))^\top$. Also, in the following proof, we will omit the superscript $w$ for notational convenience.

Due to (6.2), we have, for $t = 1, 2, \ldots,$

$$\begin{aligned}
\boldsymbol{z}(t) &= \frac{1}{\beta}\mathrm{diag}\,(\boldsymbol{\pi})^{-1}\boldsymbol{g}(t-1) + \boldsymbol{P}\boldsymbol{z}(t-1) \\
&= \frac{1}{\beta}\sum_{s=1}^{t}\boldsymbol{P}^{s-1}\mathrm{diag}\,(\boldsymbol{\pi})^{-1}\boldsymbol{g}(t-s).
\end{aligned} \tag{6.13}$$

So,

$$z_i(t) = \frac{1}{\beta}\sum_{s=1}^{t} \boldsymbol{e}_i^\top \boldsymbol{P}^{s-1}\mathrm{diag}\left(\boldsymbol{\pi}\right)^{-1}\boldsymbol{g}(t-s), \quad \text{and}$$

$$\begin{aligned}\bar{z}(t) = \boldsymbol{\pi}^\top \boldsymbol{z}(t) &= \frac{1}{\beta}\sum_{s=1}^{t} \boldsymbol{\pi}^\top \boldsymbol{P}^{s-1}\mathrm{diag}\left(\boldsymbol{\pi}\right)^{-1}\boldsymbol{g}(t-s) \\ &= \frac{1}{\beta}\sum_{s=1}^{t} \boldsymbol{\pi}^\top \mathrm{diag}\left(\boldsymbol{\pi}\right)^{-1}\boldsymbol{g}(t-s). \end{aligned} \tag{6.14}$$

Thus,

$$\begin{aligned}&\|\bar{z}(t) - z_i(t)\| \\ &= \frac{1}{\beta}\left\|\sum_{s=1}^{t}\left(\boldsymbol{\pi}^\top - \boldsymbol{e}_i^\top \boldsymbol{P}^{s-1}\right)\mathrm{diag}\left(\boldsymbol{\pi}\right)^{-1}\boldsymbol{g}(t-s)\right\| \\ &\leq \frac{L}{\beta\pi_{\min}}\sum_{s=1}^{t}\left\|\boldsymbol{\pi}^\top - \boldsymbol{e}_i^\top \boldsymbol{P}^{s-1}\right\|_1. \end{aligned} \tag{6.15}$$

Based on the Prop. 3 of [11],

$$\left\|\boldsymbol{\pi}^\top - \boldsymbol{e}_i^\top \boldsymbol{P}^{s-1}\right\|_1 \leq \sqrt{\frac{1-\pi_i}{\pi_i}}\sigma_2^{s-1}. \tag{6.16}$$

Substitute (6.16) into (6.15), we have

$$\begin{aligned}\|\bar{z}(t) - z_i(t)\| &\leq \frac{L}{\beta\pi_{\min}}\sum_{s=1}^{t}\sqrt{\frac{1-\pi_i}{\pi_i}}\sigma_2^{s-1} \\ &\leq \frac{L}{\beta\pi_{\min}}\sqrt{\frac{1-\pi_i}{\pi_i}}\frac{1}{1-\sigma_2(\boldsymbol{P})}, \end{aligned}$$

which completes the proof. □

Finally, we are ready to present the convergence theorem by combining Lemma 4 and Lemma 5.

**Theorem 1.** *Consider the sequences $\{\boldsymbol{x}_i(t)\}$ and $\{\boldsymbol{z}_i(t)\}$ generated by random-walk distributed dual averaging (Algorithm 2). Define the running average at each node $i$ as $\widehat{\boldsymbol{x}}_i(T) = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{x}_i(t)$. Then for any $\boldsymbol{x}^\star \in \mathcal{X}$ with $\|\boldsymbol{x}^\star\| \leq R$, and for each node $i \in [n]$, one has*

$$\bar{f}\left(\widehat{\boldsymbol{x}}_i\left(T\right)\right) - \bar{f}(\boldsymbol{x}^\star) \leq \frac{2LR}{\sqrt{nT(1-\sigma_2(\boldsymbol{P})}\pi_{\min}^{3/4}} \tag{6.20}$$

*when the step size $\alpha(t)$ is chosen as $\frac{\beta(\pi_{\min})^{3/4}\sqrt{1-\sigma_2(\boldsymbol{P})}R}{4L\sqrt{n}}\cdot\frac{1}{\sqrt{t}}$.*

*Proof.* Let us choose $\alpha(t)$ in the form of $c/\sqrt{t}$, where $c$ is to be optimized later.

Plugging (6.12) into (6.4), we reach

$$\begin{aligned}&\bar{f}\left(\widehat{\boldsymbol{x}}_i\left(T\right)\right) - \bar{f}(\boldsymbol{x}^\star) \\ &\leq \frac{L^2 n}{\beta\sqrt{T}}\cdot c + \frac{\beta}{2n\sqrt{T}}R^2\cdot\frac{1}{c} + \frac{6L^2}{\sqrt{T}\beta\pi_{\min}^{3/2}(1-\sigma_2(\boldsymbol{P}))}\cdot c \\ &\leq \frac{7L^2}{\sqrt{T}\beta\pi_{\min}^{3/2}(1-\sigma_2(\boldsymbol{P}))}\cdot c + \frac{\beta}{2n\sqrt{T}}R^2\cdot\frac{1}{c}, \end{aligned} \tag{6.21}$$

where we have used the fact that $\sum_{t=1}^{T} t^{-1/2} \leq \int_{t=0}^{T} t^{-1/2}dt = \sqrt{T}$.

The claimed result holds directly as we optimize the upper bound (6.21) with respect to the parameter $c$. □

**Algorithm 3** Generalized Iron Distributed Dual Averaging Method

**Input:** a predetermined nonnegative nonincreasing sequence $\{\alpha(t)\}$.
**Initialization:** $\boldsymbol{x}_i(0) = \boldsymbol{z}_i(0) = \mathbf{0}$, for all $i \in [n]$.
**for** $t = 0, 1, 2, \ldots,$ **do**

1. Stochastic subgradient calculation:

$$\mathbb{E}\left[\boldsymbol{g}_i(t)\right] \in \partial f_i(\boldsymbol{x}_i(t)), \quad \text{for each agent } i. \tag{6.17}$$

2. Dual updates:

$$\boldsymbol{z}_i(t+1) = \frac{\sum_{j \in \mathcal{N}(i) \cup \{i\}} \boldsymbol{z}_j(t) + \boldsymbol{g}_i(t)}{|\mathcal{N}(i)| + 1}, \quad \text{for each agent } i. \tag{6.18}$$

3. Primal updates:

$$\begin{aligned} \boldsymbol{x}_i(t+1) &= \text{Prox}_{t\alpha(t)\phi(\cdot)}[-\alpha(t)\boldsymbol{z}_i(t+1)] \\ &= \arg\min_{\boldsymbol{x}} \ \frac{1}{2}\left\|\boldsymbol{x} + \alpha(t)\boldsymbol{z}_i(t+1)\right\|^2 + t\alpha(t)\phi(\boldsymbol{x}), \\ &\qquad \text{for each agent } i. \end{aligned} \tag{6.19}$$

**end for**

## 7 Extension

Our random-walk distributed dual averaging method can be easily adapted to incorporate stochastic gradients to solve an optimization problem with a convex regularizer (e.g. $\ell_1$, nuclear norm). Specifically, Algorithm 3, a natural modification of the random-walk distributed dual averaging method (Algorithm 2), is capable of solving

$$\min_{\boldsymbol{x}} \quad \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x}) + \phi(\boldsymbol{x}), \tag{7.1}$$

where $\phi(\boldsymbol{x})$ is a convex regularizer. Its convergence proof follows directly from combining our analysis above and arguments used in previous literature [37, 3], which we omit here.