Convergent Adaptive Gradient Methods in Decentralized Optimization

Anonymous Authors[1]

Abstract

Adaptive gradient methods including Adam, AdaGrad, and their variants are proven to be very successful for training machine learning models such as neural nets in the past a few years. At the same time, distributed optimization is becoming increasingly popular, partly due to its success in training neural nets. With the growth of computing power and the need for using machine learning models on mobile devices, the communication cost of distributed training algorithms becomes unignorable. In response to this, more and more attention is shifted from the traditional parameter server training paradigm to decentralized training paradigms, which usually require lower communication costs. In this paper, we try to rigorously incorporate adaptive gradient methods into decentralized training, giving rise to convergent decentralized adaptive gradient methods. Specifically, we propose an algorithmic framework that can convert existing adaptive gradient methods to their decentralized counterparts. In addition, we rigorously analyze the convergence behavior of the proposed algorithmic framework and show that if an adaptive gradient method is convergent, its converted counterpart is also convergent. Finally, using the framework, we proposed the first convergent decentralized adaptive gradient method.

1. Introduction

Distributed training of machine learning models is drawing increasing attention in the past few years due to its practical benefits and necessities. Due to the evolution of computing capabilities of CPUs and GPUs, computation time in distributed training is gradually dominated by the communication time in many circumstances (Chilimbi et al., 2014; McMahan et al., 2016). In response to this fact, a large amount of recent works has been focusing on reducing communication cost for distributed training (Alistarh et al., 2017; Lin et al., 2017; Wangni et al., 2018; Stich et al., 2018; Wang et al., 2018; Tang et al., 2019). In the traditional parameter server setting where a parameter server is employed to manage communication in the whole network, many effective communication reductions have been proposed based on gradient compression and quantization. Despite these communication reduction techniques, the amount of data flow of the parameter server usually scales linearly with the number of workers. Due to this limitation, there is a rising interest in the research community in the decentralized training paradigm (Duchi et al., 2011b), where the parameter server is removed and every node only communicates with its neighbors. It has been shown in Lian et al. (2017) that decentralized training algorithms can outperform parameter server-based algorithms when the training bottleneck is the communication cost. The decentralized training paradigm is also preferred when a parameter server is not available.

In parallel to distributed training, another effective way to accelerate training is by using adaptive gradient methods like AdaGrad (Duchi et al., 2011a), Adam (Kingma & Ba, 2014) and AMSGrad (Reddi et al., 2019). Their practical benefits are proven by their popularity in training neural nets, featured by faster convergence and ease of parameter tuning compared with SGD.

Despite a large amount of literature in distributed optimization, there have been few works seriously considering bringing adaptive gradient methods into distributed training, largely due to the lack of understanding in convergence behavior of adaptive gradient methods.

In this paper, we investigate the possibility of using adaptive gradient methods in the decentralized training paradigm. Designing adaptive methods in such settings is highly nontrivial due to the already complicated update rules and the interaction between the effect of using adaptive learning rates and decentralized communication protocols.

The key result of this work is a general technique that can convert an adaptive gradient method from a centralized method to a decentralized method. More importantly, we provide a theoretical verification interface for analyzing the behavior of decentralized adaptive gradient methods converted by our technique.

By using our proposed technique, we also present a new decentralized optimization algorithm, called decentralized AMSGrad, converted by our technique from AMSGrad. Build on our proposed framework for analyzing the type of algorithms, we can characterize the convergence rate

of decentralized AMSGrad, which is the first convergent decentralized adaptive gradient method.

A novel technique in our framework is a mechanism to enforce a consensus on adaptive learning rates at different nodes. We show the importance of consensus on adaptive learning rates by proving a divergent problem instance for a recently proposed decentralized adaptive gradient method DADAM, which lacks consensus mechanisms on adaptive learning rates.

Notations: $x_{t,i}$ denotes variable x at node i and iteration t. $\|\cdot\|_{abs}$ denotes the entry-wise L_1 norm of a matrix, i.e. $\|A\|_{abs} = \sum_{i,j} A_{i,j}$. For the ease of presentation, here we also introduce some notations that will be used later in the paper.

- $G_t = [g_{t,1}, g_{t,2}, ..., g_{t,N}]$
- $M_t = [m_{t,1}, m_{t,2}, ..., m_{t,N}]$
- $X_t = [x_{t,1}, x_{t,2}, ..., x_{t,N}]$
- $\overline{\nabla f}(X_t) = \frac{1}{N}\sum_{i=1}^N \nabla f_i(x_{t,i})$
- $U_t = [u_{t,1}, u_{t,2}, ..., u_{t,N}]$
- $\tilde{U}_t = [\tilde{u}_{t,1}, \tilde{u}_{t,2}, ..., \tilde{u}_{t,N}]$
- $V_t = [v_{t,1}, v_{t,2}, ..., v_{t,N}]$
- $\hat{V}_t = [\hat{v}_{t,1}, \hat{v}_{t,2}, ..., \hat{v}_{t,N}]$
- $\overline{X}_t = \frac{1}{N}\sum_{i=1}^N x_{t,i}$
- $\overline{U}_t = \frac{1}{N}\sum_{i=1}^N u_{t,i}$
- $\overline{\tilde{U}_t} = \frac{1}{N}\sum_{i=1}^N \tilde{u}_{t,i}$

Also, we will introduce a $N \times N$ matrix W later in the paper, we denote λ_i to be its ith largest eigenvalue and define $\lambda \triangleq \max(|\lambda_2|, |\lambda_N|)$.

2. Related work

Decentralized optimization: Decentralized optimization has a long history, traditional decentralized optimization methods include well-know algorithms such as ADMM (Boyd et al., 2011), dual averaging (Duchi et al., 2011b), distributed subgradient descent (Nedic & Ozdaglar, 2009). More recent algorithms include Extra (Shi et al., 2015), Next (Di Lorenzo & Scutari, 2016) and Prox-PDA (Hong et al., 2017). While these algorithms were commonly used in applications other than deep learning, recent algorithmic advances in the machine learning community have shown that decentralized optimization can be useful for training neural nets. Lian et al. (2017) showed that a stochastic version of decentralized subgradient descent can outperform parameter server-based algorithms when the communication cost is high. (Tang et al., 2018) proposed D^2 that improves the convergence rate over stochastic subgradient descent. (Assran et al., 2018) proposed the Stochastic Gradient Push that is more robust to network failures for training neural nets. The study of decentralized training in the machine learning community is only at its initial stage. No one has seriously considered designing adaptive gradient methods in the setting of decentralized training until the recent work Nazari et al. (2019), a decentralized version of AMSGrad (Reddi et al., 2019) is proposed in Nazari et al. (2019) and it is proven to satisfy some non-standard regret.

Adaptive gradient methods: Adaptive gradient methods are popularized in recent years due to their superior performance in training neural nets. The type of methods usually refers to AdaGrad (Duchi et al., 2011a), Adam (Kingma & Ba, 2014), and their variants. Key features of such methods include the use of momentum and adaptive learning rates (which means the learning rate is changing during optimization and the learning rates on different coordinates might be different). The most adaptive gradient is Adam, which is believed to converge until the recent work Reddi et al. (2019) pointed out an error in the convergence analysis of Adam. Since then, many research efforts in the community are investigated into analyzing the convergence behavior of adaptive gradient methods. Ward et al. (2018); Li & Orabona (2018) analyzed convergence of a variant of AdaGrad without coordinate-wise learning rates. Chen et al. (2018) analyzed the convergence behavior of a broad class of algorithms including AMSGrad (Reddi et al., 2019) and AdaGrad. Zou & Shen (2018) provided a unified convergence analysis for AdaGrad with momentum. A few recent adaptive gradient methods can be found in Agarwal et al. (2018); Luo et al. (2019); Zaheer et al. (2018).

3. Decentralized training and divergence of DADAM

3.1. Decentralized optimization

In distributed optimization (with N nodes), we aim at solving the following problem

$$\min_x \frac{1}{N}\sum_{i=1}^N f_i(x) \tag{1}$$

where f_i is only accessible by the ith node. For neural net training, f_i can be viewed as the average loss of data located at node i.

Throughout the paper, we make the following assumptions for analyzing the convergence behavior of different algorithms.

Assumptions

A1: f_i's are differentiable and the gradients is L-Lipschitz, i.e. $\|\nabla f_i(x) - \nabla f_i(y)\| \le L\|x - y\|, \forall x, y$.

A2: We assume at iteration t, node i can access a stochastic gradient $g_{t,i}$. In addition, the stochastic gradients have bounded L_∞ norm and the gradients of f_i are also bounded, i.e. $\|g_{t,i}\| \le G_\infty$, $\|\nabla f_i(x)\|_\infty \le G_\infty$.

A3: The gradient estimators are unbiased and each coordinate have bounded variance, i.e. $\mathbb{E}[g_{t,i}] = \nabla f_i(x_{t,i})$ and $\mathbb{E}[([g_{t,i} - f_i(x_{t,i})]_j)^2] \le \sigma^2, \forall t, i, j$.

The assumptions A1 and A3 are standard in distributed optimization. A2 is a little stronger than the traditional assumption that the estimator has bounded variance, it is commonly used in analyses for adaptive gradient methods (Chen et al., 2018; Ward et al., 2018). One thing that should be noted is that the bounded gradient estimator assumption in A2 implies the bounded variance assumption in A3, we denote the variance bound and the estimator bound differently to avoid confusion when we use them for different purposes.

In decentralized optimization, the nodes are connected as a graph and each node only communicates to its neighbors. In such cases, one usually construct a matrix W for information sharing when designing algorithms. As can be expected, W cannot be arbitrary, the key properties required for W are listed in A4.

A4: Assumptions on matrix W.

1). $\sum_{j=1}^N W_{i,j} = 1$, $\sum_{i=1}^N W_{i,j} = 1$, $W_{i,j} \geq 0$.

2). Denote λ_i to be ith largest eigenvalue of W, we have $\lambda_1 = 1$, $|\lambda_2| < 1$, $|\lambda_N| < 1$.

3). $W_{i,j} = 0$ if node i and node j are not neighbors.

Throughout this paper, we will assume A1-A4 hold.

3.2. Divergence of DADAM

Recently, Nazari et al. (2019) initiated a trial to bring adaptive gradient methods into decentralized optimization, the resulting algorithm is DADAM, which is shown in Algorithm 1.

Algorithm 1 DADAM(with N nodes)

Input: learning rate α, current point X_t, $u_{\frac{1}{2},i} = \hat{v}_{0,i} = \epsilon \mathbf{1}, \forall i$, $m_0 = 0$ mixing matrix W
for $t = 1, 2, ..., T$ **do**
  $g_{t,i} \leftarrow \nabla f_i(x_{t,i}) + \xi_{t,i}$
  $m_{t,i} = \beta_1 m_{t-1,i} + (1-\beta_1) g_{t,i}$
  $v_{t,i} = \beta_2 v_{t-1,i} + (1-\beta_2) g_{t,i}^2$
  $\hat{v}_{t,i} = \beta_3 \hat{v}_{t,i} + (1-\beta_3) \max(\hat{v}_{t-1,i}, v_{t,i})$
  $x_{t+\frac{1}{2},i} = \sum_{j=1}^N W_{ij} x_{t,j}$
  $x_{t+1,i} = x_{t+\frac{1}{2},i} - \alpha \frac{m_{t,i}}{\sqrt{\hat{v}_{t,i}}}$
end for

DADAM is essentially a decentralized version of AMSGrad and the key modification is the use of a consensus step on optimization variable x to transmit information across the network, encouraging convergence. The matrix W is a doubly stochastic matrix (which satisfies A4) for achieving average consensus of x. Introducing such a mixing matrix is a standard approach for decentralizing an algorithm, such as distributed gradient descent (Nedic & Ozdaglar, 2009; Yuan et al., 2016). It is proven in Nazari et al. (2019) that DADAM admits a non-standard regret bound in the online setting, however, whether the algorithm can converge to stationary points in standard offline settings such training neural nets is still unknown.

In the following, we show the DADAM may fail to converge in offline nonconvex optimization settings.

Theorem 1. *There exist a problem satisfying $A1-A4$ where DADAM fail to converge.*

Proof: Consider a 1 dimensional optimization problem distributed onto two nodes

$$\min_x \frac{1}{2} \sum_{i=1}^2 f_i(x) \tag{2}$$

where $f_i(x) = \frac{1}{2}(x - a_i)^2$ and $a_1 = 0$, $a_2 = 1$.

The network contains only two nodes and the matrix W satisfy $W_{ij} = \frac{1}{2}, \forall i, j$.

We consider running DADAM with $\beta_1 = \beta_2 = \beta_3 = 0$ and $\epsilon = 0.6$ for simplicity. Suppose we initialize DADAM at $x_{1,i} = 0, \forall i$ and use learning rate $\alpha = 0.001$. We have at $x_{1,i} = 0$, $\nabla f_1(x_{1,1}) = 0$, $\nabla f_2(x_{1,2}) = 1$, this will lead to $\hat{v}_{1,1} = 0.6$ and $\hat{v}_{1,2} = 1$. Thus, from step 1, we will have $\hat{v}_{1,2} \geq 1$. In addition, it s easy proved that with the stepsize selection, we always have $\hat{v}_{1,1} < 1$, in fact, it will not even reach 0.6. Thus, in the later iterations, the gradient of losses on node 1 and 2 will be scaled differently. This scaling is equivalent running gradient descent on a objective where the losses of the two nodes are scaled by different factors. In such a case, the algorithm will converge to a stationary point of a weighted average of the loss on node 1. Since the weight of the losses on the two nodes are different and the problem is a quadratic problem with only one minimizer and the unbalanced weights on the two functions yields a different minimizer, the algorithm will not converge to the unique stationary point of the original loss (which is $x = 0.5$). □

Theorem 1 says that though DADAM is proven to satisfy some regret bounds (Nazari et al., 2019), it can fail to converge to stationary points in the nonconvex offline setting, which a common setting for training neural nets. We conjecture that this inconsistency is due to the definition of the regret in Nazari et al. (2019). In the next section, we will design decentralized adaptive gradient methods that are guaranteed to converge to stationary points.

4. Convergent decentralized adaptive gradient methods

In this section, we will discuss difficulties of designing adaptive gradient methods in decentralized optimization and introduce an algorithmic framework that will convert existing convergent adaptive gradient methods to their decentral-

ized counterparts. By using the framework, we proposed the first convergent decentralized adaptive gradient method, converted from AMSGrad.

4.1. Importance and difficulties of consensus on adaptive learning rates

The divergent example in the previous section implies that we should synchronize the adaptive learning rates on different nodes. This can be easy to achieve in the parameter server setting where all the nodes are sending their gradients to a parameter server at each iteration. The parameter server can use the received gradients to maintain a sequence of synchronized adaptive learning rates when updating the parameters. However, in the situation of decentralized training, every node can only communication with its neighbors and a parameter server does not exist. Since every node can only communicate with its neighbors, the information for updating the adaptive learning rates can be only shared locally instead of broadcasted over the whole network, this make it impossible to obtain an synchronized updated adaptive learning rate in a single iteration using all the information in the network.

One way to solve this problem is to design communication protocols to give each node access to the same aggregated gradients over the whole network at least periodically if not at every iteration, so that the nodes can update their individual adaptive learning rates based on the same information to generate a synchronized sequence of adaptive learning rates. However, such a solution will introduce a significant amount of extra communication cost since it involves broadcasting over the network. Also, this is more of a system level solution instead of a algorithmic level solution.

Another way to solve this problem is by letting the sequences of adaptive learning rates on different nodes consent gradually, as the number of iteration grows. Intuitively, if the adaptive learning rates can consent fast enough, the difference among the adaptive learning rates on different nodes will not affect the convergence of the algorithm. The benefit of such an approach is that we do not need to introduce too much extra communication cost and as we will show later, it will produce an framework that automatically convert existing adaptive gradient methods to their decentralized counterparts. Yet, the benefits do not come for free. One need to design a way to ensure consensus of adaptive learning rates and this procedure should have a relatively low cost and be easy to implement. More importantly, such a design will further complicates the already convoluted convergence analysis of adaptive gradient methods.

In the next section, we will introduce our designed algorithmic framework based on this approach and its theoretical guarantee.

4.2. On decentralized adaptive gradient methods

As mentioned before, we need to choose a method to implement consensus of adaptive learning rates and there are many ways to do this. While each node can have different $\hat{v}_{t,i}$ in DADAM, one can keep track of the min/max/average of these adaptive learning rates and use the tracked quantity to update the adaptive learning rates. Also one can predefined some convergent lower and upper bounds to gradually synchronize the adaptive learning rates on different nodes like what the authors did for AdaBound (Luo et al., 2019). We choose to use average consensus on $\hat{v}_{t,i}$ because in adaptive gradient methods such as AdaGrad and Adam, $\hat{v}_{t,i}$ approximate the second moment of gradient estimator, the average of estimations of second moments from different nodes is an estimation of second moment on the whole network. Also, this design will not introduce any extra tunable parameters that will complicates the parameter tuning process.

Algorithm 2 decentralized adaptive gradient method (with N nodes)

Input: learning rate α, initial point $x_{1,i} = x_{init}, u_{\frac{1}{2},i} = \hat{v}_{0,i}, m_{0,i} = 0, \forall i$, mixing matrix W
for $t = 1, 2, ..., T$ **do**
  $g_{t,i} \leftarrow \nabla f_i(x_{t,i}) + \xi_{t,i}$
  $m_{t,i} = \beta_1 m_{t-1,i} + (1-\beta_1) g_{t,i}$
  $\hat{v}_{t,i} = r_t(g_{1,i}, ..., g_{t-1,i})$
  $x_{t+\frac{1}{2},i} = \sum_{j=1}^{N} W_{ij} x_{t,j}$
  $\tilde{u}_{t,i} = \sum_{j=1}^{N} W_{ij} \tilde{u}_{t-\frac{1}{2},j}$
  $u_{t,i} = \max(\tilde{u}_{t,i}, \epsilon)$
  $x_{t+1,i} = x_{t+\frac{1}{2},i} - \alpha \frac{m_{t,i}}{\sqrt{u_{t,i}}}$
  $\tilde{u}_{t+\frac{1}{2},i} = \tilde{u}_{t,i} - \hat{v}_{t-1,i} + \hat{v}_{t,i}$
end for

Theorem 2 presents the convergence guarantee of Algorithm 2.

Theorem 2. *Assume* $\|g_{t,i}\|_\infty \le G_\infty$, $\|\nabla f_i(x)\|_\infty \le G_\infty$ *and set* $\alpha = 1/\sqrt{Td}$. *When* $\alpha \le \frac{\epsilon^{0.5}}{16L}$, *Algorithm 2 yields the following regret bound*

$$\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\left\|\frac{\nabla f(\overline{X}_t)}{\overline{U}_t^{1/4}}\right\|^2\right] \\
\le& C_1\frac{\sqrt{d}}{\sqrt{T}}\left(\mathbb{E}[f(Z_1)] - \min_z f(z) + \frac{\sigma^2}{N}\right) + \frac{C_2}{T} + \frac{C_3}{T^{1.5}d^{0.5}} \\
&+ \left(\frac{C_4}{TN^{0.5}} + \frac{C_5}{T^{1.5}d^{0.5}N^{0.5}}\right)\mathbb{E}\left[\sum_{t=1}^{T}\|(-\hat{V}_{t-2} + \hat{V}_{t-1})\|_{abs}\right]
\end{aligned} \tag{3}$$

where $\|\cdot\|_{abs}$ *denotes the entry-wise* L_1 *norm of a matrix (i.e* $\|A\|_{abs} = \sum_{i,j}|A_{ij}|$*) and* C_1, C_2, C_3, C_4, C_5 *are defined*

as

$$
\begin{aligned}
C_1 =& \max(4, 4L/\epsilon) \\
C_2 =& 6\left(\left(\frac{\beta_1}{1-\beta_1}\right)^2 + \left(\frac{1}{1-\lambda}\right)^2\right) L\frac{G_\infty^2}{\epsilon^{1.5}} \\
C_3 =& 16L^2\left(\frac{1}{1-\lambda}\right)\frac{G_\infty^2}{\epsilon^2} \\
C_4 =& \frac{2}{\epsilon^{1.5}}\frac{1}{1-\lambda}\left(\lambda + \frac{\beta_1}{1-\beta_1}\right) G_\infty^2 \\
C_5 =& \frac{2}{\epsilon^2}\frac{1}{1-\lambda} L\left(\frac{\beta_1}{1-\beta_1}\right)^2 G_\infty^2 + \frac{4}{\epsilon^2}\frac{\lambda}{1-\lambda} L G_\infty^2 \qquad (4)
\end{aligned}
$$

which are constants independent of d, T and N.

Proof: The proof sketch can be found in Section 5 and the complete proof can be found in Appendix A.1.

Remark: From the theorem, it can be seen that if $\mathbb{E}\left[\sum_{t=1}^T \|(-\hat{V}_{t-2} + \hat{V}_{t-1})\|_{abs}\right] = o(T)$ and $\bar{U}_t$ is upper bounded, the algorithm is guaranteed to converge to stationary points. Intuitively, this says that if the adaptive learning rates on different nodes do not change too fast, the algorithm can converge. This is intuitively true as in Chen et al. (2018), it is shown that if such a condition is violated, an algorithm can diverge. Furthermore, the theorem shows the benefit of using more nodes. As N becomes larger, the term σ^2/N will be small, this is also justified by intuition that with the growth of N, the training process tends to be more stable.

In the following, we will present a notable special case of our algorithmic framework, decentralized AMSGrad, which is a decentralized variant of AMSGrad in our framework.

Algorithm 3 decentralized AMSGrad (with N nodes)

Input: learning rate α, initial point $x_{1,i} = x_{init}, u_{\frac{1}{2},i} = \hat{v}_{0,i} = \epsilon\mathbf{1}$ (with $\epsilon \geq 0$), $m_{0,i} = 0, \forall i$, mixing matrix W
for $t = 1, 2, ..., T$ **do**
  $g_{t,i} \leftarrow \nabla f_i(x_{t,i}) + \xi_{t,i}$
  $m_{t,i} = \beta_1 m_{t-1,i} + (1-\beta_1) g_{t,i}$
  $v_{t,i} = \beta_2 v_{t-1,i} + (1-\beta_2) g_{t,i}^2$
  $\hat{v}_{t,i} = \max(\hat{v}_{t-1,i}, v_{t,i})$
  $x_{t+\frac{1}{2},i} = \sum_{j=1}^N W_{ij} x_{t,j}$
  $\tilde{u}_{t,i} = \sum_{j=1}^N W_{ij} \tilde{u}_{t-\frac{1}{2},j}$
  $u_{t,i} = \max(\tilde{u}_{t,i}, \epsilon)$
  $x_{t+1,i} = x_{t+\frac{1}{2},i} - \alpha \frac{m_{t,i}}{\sqrt{u_{t,i}}}$
  $\tilde{u}_{t+\frac{1}{2},i} = \tilde{u}_{t,i} - \hat{v}_{t-1,i} + \hat{v}_{t,i}$
end for

Compared with DADAM, the above algorithm uses a dynamic average consensus mechanism to keep track of average of $\{\hat{v}_{t,i}\}_{i=1}^N$, stored as $\tilde{u}_{t,i}$ on ith node, and uses $u_{t,i} = \max(\tilde{u}_{t,i}, \epsilon)$ for updating the adaptive learning rate for ith node. As the number of iteration grows, even though $\hat{v}_{t,i}$ on different nodes can converge to different constants, all the $u_{t,i}$ will be converge to the same number $\lim_{t\to\infty} \frac{1}{N}\sum_{i=1}^N \hat{v}_{t,i}$ if the limit exists. The use of this average consensus mechanism enables the consensus of adaptive learning rates on different nodes, which consequentially guarantees convergence to stationary points. The consensus of adaptive learning rates is the key difference between decentralized AMSGrad and DADAM and is the reason why decentralized AMSGrad is a convergent algorithm while DADAM is not.

The following theorem presents the convergent guarantee of Algorithm 3.

Theorem 3. *Assume $\|g_{t,i}\|_\infty \leq G_\infty$, $\|\nabla f_i(x)\|_\infty \leq G_\infty$ and set $\alpha = 1/\sqrt{Td}$. When $\alpha \leq \frac{\epsilon^{0.5}}{16L}$, Algorithm 3 yields the following regret bound*

$$
\begin{aligned}
&\frac{1}{T}\sum_{t=1}^T \mathbb{E}\left[\left\|\frac{\nabla f(\overline{X}_t)}{\overline{U}_t^{1/4}}\right\|^2\right] \\
\leq& C_1'\frac{\sqrt{d}}{\sqrt{T}}\left(\mathbb{E}[f(Z_1)] - \min_z f(z) + \frac{\sigma^2}{N}\right) + \frac{C_2'}{T} + \frac{C_3'}{T^{1.5}d^{0.5}} \\
&+ \frac{d}{T}\sqrt{N}C_4' + \frac{\sqrt{d}}{T^{1.5}}\sqrt{N}C_5' \qquad (5)
\end{aligned}
$$

where

$$
\begin{aligned}
&C_1' = C_1,\ C_2' = C_2,\ C_3' = C_3, \\
&C_4' = C_4 G_\infty^2,\ C_5' = C_5 G_\infty^2 \qquad (6)
\end{aligned}
$$

and C_1, C_2, C_3, C_4, C_5 are constants independent of d, T and N defined in Theorem 2.

Proof: See Appendix A.2

Remark: The above theorem says that Algorithm 3 converges with a rate of $O(\sqrt{d}/\sqrt{T})$ when T is large, which is the best known convergence rate under the given assumptions. Note that in some literature, SGD admits a convergence rate of $O(1/\sqrt{T})$ without any dimension dependency, such an improved convergence rate is under the assumption that the gradient estimator have bounded L_2 norm, which can hide a dimension dependency of $\sqrt{d}$ in the final convergence rate. One can

In the next section, we will present the proof sketch of Theorem 2 since the whole proof is complicated and the convergence analysis is one of our main contributions.

5. Proof Sketch of Theorem 2

In this section, we will present the proof sketch for convergence analysis of Algorithm 2 (proof sketch of Theorem 2).

To prove convergence of the algorithm, we need to first define an auxiliary sequence of iterates

$$
Z_t = \overline{X}_t + \frac{\beta_1}{1-\beta_1}(\overline{X}_t - \overline{X}_{t-1}) \qquad (7)
$$

with $\overline{X}_0 \triangleq \overline{X}_1$.

Such an auxiliary sequence can help us deal with the bias brought by the momentum and simplifies the convergence analysis. Similar constructions are also used for analyzing SGD with momentum and centralized adaptive gradient methods (Yan et al., 2018; Chen et al., 2018).

One can then prove that the auxiliary sequence Z_t follows the update rule

$$\begin{aligned} Z_{t+1} - Z_t =& \alpha \frac{\beta_1}{1-\beta_1} \frac{1}{N} \sum_{i=1}^N m_{t-1,i} \odot \left(\frac{1}{\sqrt{u_{t-1,i}}} - \frac{1}{\sqrt{u_{t,i}}}\right) \\ & - \alpha \frac{1}{N} \sum_{i=1}^N \frac{g_{t,i}}{\sqrt{u_{t,i}}}. \end{aligned} \tag{8}$$

The above update rule is desired because momentum does not appear in the quantity $\frac{1}{N}\sum_{i=1}^N \frac{g_{t,i}}{\sqrt{u_{t,i}}}$ and this simplification is helpful because it is directly related to the current gradients instead of the exponential average of past gradients.

Then by smoothness, we have the basic inequality in non-convex optimization

$$f(Z_{t+1}) \le f(Z_t) + \langle \nabla f(Z_t), Z_{t+1} - Z_t \rangle + \frac{L}{2}\|Z_{t+1} - Z_t\|^2 \tag{9}$$

which will translate into

$$\begin{aligned} & \alpha \mathbb{E}\left[\left\langle \nabla f(Z_t), \frac{1}{N}\sum_{i=1}^N \frac{\nabla f_i(x_{t,i})}{\sqrt{u_{t,i}}} \right\rangle\right] \\ \le & \mathbb{E}[f(Z_t)] - \mathbb{E}[f(Z_{t+1})] + \frac{L}{2}\mathbb{E}\left[\|Z_{t+1} - Z_t\|^2\right] \\ & + \alpha \frac{\beta_1}{1-\beta_1} \mathbb{E}\left[\left\langle \nabla f(Z_t), \frac{1}{N}\sum_{i=1}^N \left(\frac{m_{t-1,i}}{\sqrt{u_{t-1,i}}} - \frac{m_{t-1,i}}{\sqrt{u_{t,i}}}\right)\right\rangle\right] \end{aligned} \tag{10}$$

after taking expectation and rearranging.

As in almost all convergence analyses, we will need to convert the LHS of the above inequality to the norm of gradients and upper bound all the quantities on the RHS to derive its convergence rate. We will do so now.

We use following inequality to split out the norm of gradients from LHS of (10)

$$\begin{aligned} & \left\langle \nabla f(Z_t), \frac{1}{N}\sum_{i=1}^N \frac{\nabla f_i(x_{t,i})}{\sqrt{\overline{U}_t}} \right\rangle \\ \ge & \frac{1}{2}\left\| \frac{\nabla f(\overline{X}_t)}{\overline{U}_t^{1/4}} \right\|^2 - \frac{3}{2}\left\| \frac{\nabla f(Z_t) - \nabla f(\overline{X}_t)}{\overline{U}_t^{1/4}} \right\|^2 \\ & - \frac{3}{2}\left\| \frac{\frac{1}{N}\sum_{i=1}^N \nabla f_i(x_{t,i}) - \nabla f(\overline{X}_t)}{\overline{U}_t^{1/4}} \right\|^2 \end{aligned} \tag{11}$$

Then summing over t from 1 to T and do some simple algebraic manipulations, we can get

$$\begin{aligned} & \frac{1}{T}\sum_{t=1}^T \mathbb{E}\left[\left\| \frac{\nabla f(\overline{X}_t)}{\overline{U}_t^{1/4}} \right\|^2\right] \\ \le & \frac{2}{T\alpha}(\mathbb{E}[f(Z_1)] - \mathbb{E}[f(Z_{T+1})]) + \frac{L}{T\alpha}\sum_{t=1}^T \mathbb{E}\left[\|Z_{t+1} - Z_t\|^2\right] \\ & + \frac{2}{T}\frac{\beta_1}{1-\beta_1} \underbrace{\sum_{t=1}^T \mathbb{E}\left[\left\langle \nabla f(Z_t), \frac{1}{N}\sum_{i=1}^N \left(\frac{m_{t-1,i}}{\sqrt{u_{t-1,i}}} - \frac{m_{t-1,i}}{\sqrt{u_{t,i}}}\right)\right\rangle\right]}_{T_1} \\ & + \underbrace{\frac{2}{T}\sum_{t=1}^T \mathbb{E}\left[\left\langle \nabla f(Z_t), \frac{1}{N}\sum_{i=1}^N \left(\frac{\nabla f_i(x_{t,i})}{\sqrt{\overline{U}_t}} - \frac{\nabla f_i(x_{t,i})}{\sqrt{u_{t,i}}}\right)\right\rangle\right]}_{T_2} \\ & \left.\begin{aligned} & + \tfrac{3}{T}\textstyle\sum_{t=1}^T \mathbb{E}\left[\left\| \frac{\frac{1}{N}\sum_{i=1}^N \nabla f_i(x_{t,i}) - \nabla f(\overline{X}_t)}{\overline{U}_t^{1/4}} \right\|^2\right] \\ & + \tfrac{3}{T}\textstyle\sum_{t=1}^T \mathbb{E}\left[\left\| \frac{\nabla f(Z_t) - \nabla f(\overline{X}_t)}{\overline{U}_t^{1/4}} \right\|^2\right] \end{aligned} \right\} T_3. \end{aligned} \tag{12}$$

Then we need to upper bound T_1, T_2, T_3 and the quadratic term of $Z_{t+1} - Z_t$.

We first bound T_3 since it is the most complicated term. By using smoothness again, it can be upper bound by quantities proportional to

$$\sum_{t=1}^T \left\|Z_t - \overline{X}_t\right\|^2 \text{ and } \sum_{t=1}^T \frac{1}{N}\sum_{i=1}^N \left\|x_{t,i} - \overline{X}_t\right\|^2. \tag{13}$$

While the former quantity can be bounded as

$$\sum_{t=1}^T \left\|Z_t - \overline{X}_t\right\|^2 \le T\left(\frac{\beta_1}{1-\beta_1}\right)^2 \alpha^2 d \frac{G_\infty^2}{\epsilon} \tag{14}$$

using update rule of X_t and definition of Z_t, the later quantity requires a more complicated analysis which yields

$$\sum_{t=1}^T \frac{1}{N}\sum_{i=1}^N \left\|x_{t,i} - \overline{X}_t\right\|^2 \le T\alpha^2 \left(\frac{1}{1-\lambda}\right)^2 d G_\infty^2 \frac{1}{\epsilon} \tag{15}$$

where $\lambda = \max(|\lambda_2|, |\lambda_N|)$ and recall that λ_i is ith largest eigenvalue of W.

With the above inequalities, we can get an upper bound for T_3 proportional to α^2, when α is small, this will be a small quantity (as will be evident later, the dominant quantity is in the order of α).

Now we need to look at T_2, after some derivation, this term can be upper bounded by

$$T_2 \le \frac{G_\infty^2}{N}\mathbb{E}\left[\sum_{t=1}^T \frac{1}{2\epsilon^{1.5}}\| - \sum_{l=2}^N \tilde{U}_t q_l q_l^T \|_{abs}\right] \tag{16}$$

where q_l is the eigenvector corresponding to lth largest eigenvalue of W and $\|\cdot\|_{abs}$ is the entry-wise L_1 norm of matrices.

In addition, after going through some further analysis, we can show that

$$\sum_{t=1}^{T}\|-\sum_{l=2}^{N}\tilde{U}_t q_l q_l^T\|_{abs}\leq\sqrt{N}\sum_{o=0}^{T-1}\frac{\lambda}{1-\lambda}\|(-\hat{V}_{o-1}+\hat{V}_o)\|_{abs} \tag{17}$$

which can finally show that T_2 admits an upper bound proportional to $\sum_{o=0}^{T-1}\|(-\hat{V}_{o-1}+\hat{V}_o)\|_{abs}$.

As for T_1, using some similar techniques for bounding T_2, we can also upper bound it as a quantity proportional to $\sum_{o=0}^{T-1}\|(-\hat{V}_{o-1}+\hat{V}_o)\|_{abs}$, which is

$$T_1\leq G_\infty^2\frac{1}{2\epsilon^{1.5}}\frac{1}{\sqrt{N}}\mathbb{E}\left[\frac{1}{1-\lambda}\sum_{t=1}^{T}\|(-\hat{V}_{t-2}+\hat{V}_{t-1})\|_{abs}\right] \tag{18}$$

What remains is to upper bound $\sum_{t=1}^{T}\mathbb{E}\left[\|Z_{t+1}-Z_t\|^2\right]$ in (12). Upper bounding this term is another difficult task because we want to show the scaling of variance with respect to N and the variance is hidden in the quantity.

To bound this term, we first show that

$$\begin{aligned}\|Z_{t+1}-Z_t\|^2\leq&2\alpha^2\left(\frac{\beta_1}{1-\beta_1}\right)^2G_\infty^2\frac{1}{N}\frac{1}{2\epsilon^2}\|\tilde{U}_t-\tilde{U}_{t-1}\|_{abs}\\&+2\alpha^2\left\|\frac{1}{N}\sum_{i=1}^{N}\frac{g_{t,i}}{\sqrt{u_{t,i}}}\right\|^2\end{aligned} \tag{19}$$

with some simple derivations.

Then we can prove

$$\begin{aligned}&\sum_{t=1}^{T}\mathbb{E}[\|Z_{t+1}-Z_t\|^2]\\\leq&\alpha^2\left(\frac{\beta_1}{1-\beta_1}\right)^2\frac{G_\infty^2}{\sqrt{N}}\frac{1}{\epsilon^2}\frac{1}{1-\lambda}\mathbb{E}\left[\sum_{t=1}^{T}\|(-\hat{V}_{t-2}+\hat{V}_{t-1})\|_{abs}\right]\\&+2\alpha^2\sum_{t=1}^{T}\mathbb{E}\left[\left\|\frac{1}{N}\sum_{i=1}^{N}\frac{g_{t,i}}{\sqrt{u_{t,i}}}\right\|^2\right]\end{aligned} \tag{20}$$

with techniques we used for bounding T_1.

Then we can split out the variance from the above quantities as

$$\mathbb{E}\left[\left\|\frac{1}{N}\sum_{i=1}^{N}\frac{g_{t,i}}{\sqrt{u_{t,i}}}\right\|^2\right]\leq\mathbb{E}\left[\left\|\frac{1}{N}\sum_{i=1}^{N}\frac{\nabla f_i(x_{t,i})}{\sqrt{u_{t,i}}}\right\|^2\right]+\frac{d}{N}\frac{\sigma^2}{\epsilon} \tag{21}$$

with a few steps of derivations.

The next step of the analysis should be merging the term $\mathbb{E}\left[\left\|\frac{1}{N}\sum_{i=1}^{N}\frac{\nabla f_i(x_{t,i})}{\sqrt{u_{t,i}}}\right\|^2\right]$ in (21) with the decent quantity, i.e. the LHS of (12). This should be easy in the analysis of SGD since the term corresponding to $\mathbb{E}\left[\left\|\frac{1}{N}\sum_{i=1}^{N}\frac{\nabla f_i(x_{t,i})}{\sqrt{u_{t,i}}}\right\|^2\right]$ in SGD is the same as the term corresponding to the LHS of (12).

While the merging cannot be done yet, we need some further analysis. We can split the term discussed before as

$$\begin{aligned}&\sum_{t=1}^{T}\mathbb{E}\left[\left\|\frac{1}{N}\sum_{i=1}^{N}\frac{\nabla f_i(x_{t,i})}{\sqrt{u_{t,i}}}\right\|^2\right]\\\leq&2\sum_{t=1}^{T}\mathbb{E}\left[\left\|\frac{1}{N}\sum_{i=1}^{N}\frac{\nabla f_i(x_{t,i})}{\sqrt{\overline{U}_t}}\right\|^2\right]\\&+2\sum_{t=1}^{T}\mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}G_\infty^2\frac{1}{\sqrt{\epsilon}}\left\|\frac{1}{\sqrt{u_{t,i}}}-\frac{1}{\sqrt{\overline{U}_t}}\right\|_1\right]\end{aligned} \tag{22}$$

and further

$$\begin{aligned}&\sum_{t=1}^{T}\mathbb{E}\left[\left\|\frac{1}{N}\sum_{i=1}^{N}\frac{\nabla f_i(x_{t,i})}{\sqrt{\overline{U}_t}}\right\|^2\right]\\\leq&2\sum_{t=1}^{T}\mathbb{E}\left[\left\|\frac{\nabla f(\overline{X}_t)}{\sqrt{\overline{U}_t}}\right\|^2\right]\\&+2\sum_{t=1}^{T}\mathbb{E}\left[\left\|\frac{1}{N}\sum_{i=1}^{N}\frac{\nabla f_i(\overline{X}_t)-\nabla f_i(x_{t,i})}{\sqrt{\overline{U}_t}}\right\|^2\right].\end{aligned} \tag{23}$$

The first term on the RHS of (23) can be easily merged with the LHS of (12) and the two terms split out can be bounded using techniques we used for bounding T_2 and T_3 which is omitted here.

Now we can combine all the bounds we derived before and this will lead to

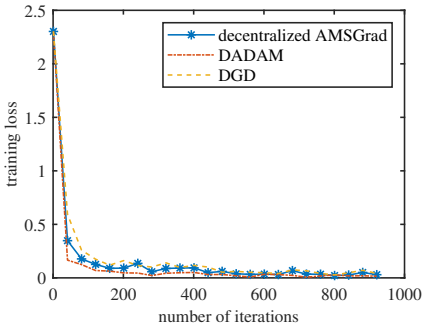
$$\begin{aligned}&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\left\|\frac{\nabla f(\overline{X}_t)}{\overline{U}_t^{1/4}}\right\|^2\right]\\\leq&\frac{2}{T\alpha}(\mathbb{E}[f(Z_1)]-\mathbb{E}[f(Z_{T+1})])+2L\alpha\frac{d}{N}\frac{\sigma^2}{\epsilon}\\&+8L\alpha\frac{1}{\sqrt{\epsilon}}\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\left\|\frac{\nabla f(\overline{X}_t)}{\overline{U}_t^{1/4}}\right\|^2\right]+8\alpha^3L^2\left(\frac{1}{1-\lambda}\right)d\frac{G_\infty^2}{\epsilon^2}\\&+3\alpha^2d\left(\left(\frac{\beta_1}{1-\beta_1}\right)^2+\left(\frac{1}{1-\lambda}\right)^2\right)L\frac{G_\infty^2}{\epsilon^{1.5}}\\&+\frac{1}{T\epsilon^{1.5}}\left(L\alpha\left(\frac{\beta_1}{1-\beta_1}\right)^2\frac{1}{\epsilon^{0.5}}+\lambda+\frac{\beta_1}{1-\beta_1}+2L\alpha\frac{1}{\epsilon^{0.5}}\lambda\right)\\&\times\frac{G_\infty^2}{\sqrt{N}}\frac{1}{1-\lambda}\mathbb{E}\left[\sum_{t=1}^{T}\|(-\hat{V}_{t-2}+\hat{V}_{t-1})\|_{abs}\right].\end{aligned} \tag{24}$$

Finally, after some further algebraic manipulations and substitutions, we can get the desired result.

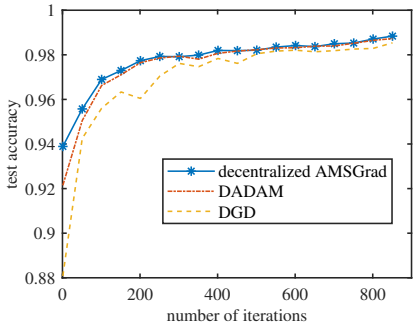
The technical details can be found in the complete proof in Appendix A.1

6. Experiments

In this section, we conduct experiments to test the performance of Algorithm 3 (decentralized AMSGrad) on both homogeneous data distribution and heterogeneous data distribution (i.e. the data generating distribution on different nodes are different). We compare it with DADAM and the decentralized stochastic gradient descent (DGD) (Lian et al., 2017). The task is training a CNN with 3 convolution layers followed by a fully connected layer on MNIST. We set $\epsilon = 1e-6$ for both decentralized AMSGrad and DADAM, the learning rate is chosen from [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6] based on validation accuracy for all algorithms. In all the experiments, the graph contains 5 nodes and the nodes form a ring, each node can only talk with its two adjacent neighbors. We set $W_{ij} = 1/3$ if there nodes i and j are neighbors and $W_{ij} = 0$ otherwise for the mixing matrix. More details and experiments can be found in Appendix A.4.



(a) Training loss

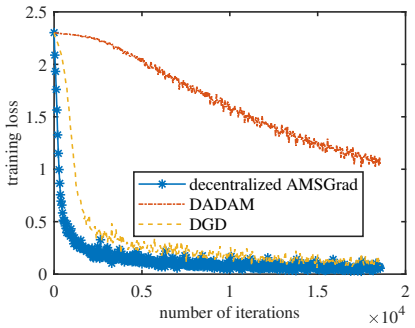


(b) Test accuracy

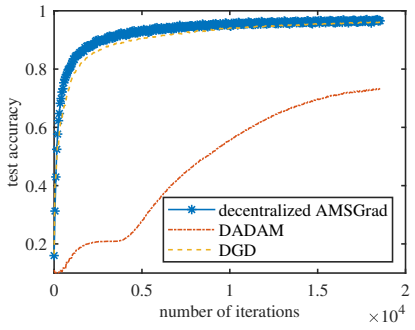
Figure 1: Performance comparison on heterogeneous data

Figure 1 shows the performance of different algorithms on homogeneous data. The whole dataset is shuffled evenly splitted to different nodes. We can see that decentralized AMSGrad and DADAM performs quite similarly and DGD is slower compared with them in terms of both training loss and test accuracy. Though we have prove in previous sections that DADAM is not a convergent algorithm, its performance is still quite good on homogeneous data. The reason is that the adaptive learning rates tend to be similar on different nodes when we have homogeneous data distribution. However, this is usually not true when we have heterogeneous data distribution. This motivates us to compare the performance of the algorithms on a different data distribution.

In Figure 2, we compare the performance of different algorithms on heterogeneous data. In this case, each node only contains training data with two labels out of ten. We can see that all algorithm converges significantly slower compared with the case with homogeneous data. Especially, the performance of DADAM deteriorates significantly. decentralized AMSGrad achieves the best training and testing performance in this experiment.



(a) Training loss



(b) Test accuracy

Figure 2: Performance comparison on heterogeneous data

7. Conclusion

This paper studies the problem of designing adaptive gradient methods for decentralized training. We proposed an algorithmic framework that can convert existing adaptive gradient methods to decentralized methods. With rigorous convergence analysis, we show that if the original algorithm satisfies some minor conditions, the converted algorithm by our framework can guarantee convergence to stationary points. By applying our framework to AMSGrad, we proposed the first convergent adaptive gradient methods. Experiments show that the proposed algorithm achieves better performance than the baselines.

References

Naman Agarwal, Brian Bullins, Xinyi Chen, Elad Hazan, Karan Singh, Cyril Zhang, and Yi Zhang. The case for full-matrix adaptive regularization. *arXiv preprint arXiv:1806.02958*, 2018.

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pp. 1709–1720, 2017.

Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Michael Rabbat. Stochastic gradient push for distributed deep learning. *arXiv preprint arXiv:1811.10792*, 2018.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

Xiangyi Chen, Sijia Liu, Ruoyu Sun, and Mingyi Hong. On the convergence of a class of adam-type algorithms for non-convex optimization. *arXiv preprint arXiv:1808.02941*, 2018.

Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 571–582, 2014.

Paolo Di Lorenzo and Gesualdo Scutari. Next: In-network nonconvex optimization. *IEEE Transactions on Signal and Information Processing over Networks*, 2(2):120–136, 2016.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12 (Jul):2121–2159, 2011a.

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, 2011b.

Mingyi Hong, Davood Hajinezhad, and Ming-Min Zhao. Prox-pda: The proximal primal-dual algorithm for fast distributed nonconvex optimization and learning over networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1529–1538. JMLR. org, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Xiaoyu Li and Francesco Orabona. On the convergence of stochastic gradient descent with adaptive stepsizes. *arXiv preprint arXiv:1805.08114*, 2018.

Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pp. 5330–5340, 2017.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*, 2019.

H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

Parvin Nazari, Davoud Ataee Tarzanagh, and George Michailidis. Dadam: A consensus-based distributed adaptive gradient method for online optimization. *arXiv preprint arXiv:1901.09109*, 2019.

Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48, 2009.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Wei Shi, Qing Ling, Gang Wu, and Wotao Yin. Extra: An exact first-order algorithm for decentralized consensus optimization. *SIAM Journal on Optimization*, 25(2):944–966, 2015.

Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. In *Advances in Neural Information Processing Systems*, pp. 4447–4458, 2018.

Hanlin Tang, Xiangru Lian, Ming Yan, Ce Zhang, and Ji Liu. D^2: Decentralized training over decentralized data. *arXiv preprint arXiv:1803.07068*, 2018.

Hanlin Tang, Xiangru Lian, Tong Zhang, and Ji Liu. Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression. *arXiv preprint arXiv:1905.05957*, 2019.

Hongyi Wang, Scott Sievert, Shengchao Liu, Zachary Charles, Dimitris Papailiopoulos, and Stephen Wright.

Atomo: Communication-efficient learning via atomic sparsification. In *Advances in Neural Information Processing Systems*, pp. 9850–9861, 2018.

Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*, pp. 1299–1309, 2018.

Rachel Ward, Xiaoxia Wu, and Leon Bottou. Adagrad stepsizes: Sharp convergence over nonconvex landscapes, from any initialization. *arXiv preprint arXiv:1806.01811*, 2018.

Yan Yan, Tianbao Yang, Zhe Li, Qihang Lin, and Yi Yang. A unified analysis of stochastic momentum methods for deep learning. *arXiv preprint arXiv:1808.10396*, 2018.

Kun Yuan, Qing Ling, and Wotao Yin. On the convergence of decentralized gradient descent. *SIAM Journal on Optimization*, 26(3):1835–1854, 2016.

Manzil Zaheer, Sashank Reddi, Devendra Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pp. 9793–9803, 2018.

Fangyu Zou and Li Shen. On the convergence of weighted adagrad with momentum for training deep neural networks. *arXiv preprint arXiv:1808.03408*, 2018.