紹介論文

DRACO: Byzantine-resilient Distributed Training via Redundant Gradients

L. Chen, Z. Charles, D. Papailiopoulos, et al

In *Proceedings of the 35th International Conference on Machine Learning*, pages 903-912, 2018
(ICML2018)

概要

分散システムは Byzantine 障害などに対して脆弱である. 最近の研究では, ロバストネスを保証するために, 集約規則として Median を取るなどの取り組みがされている. しかしながら,Median ベースの計算は大きなシステムの場合, 計算量がとても大きくなり, 収束性の保証にも強い仮定が必要となる. この論文では DARACO という手法を提案する.DARACO は冗長性を利用してロバストネスを実現する. パラメータサーバが冗長な勾配を評価することで敵対的な更新の影響を排除する. 多くの大規模なモデルの実験して,Median ベースのアプローチよりも数倍 DARACO の方が早いことを示す.

1 Introduction

並列最適化アルゴリズムは大規模なモデルに対して一般的手法となっている. しかしながら, 悪意ある攻撃やソフトウェアのエラーによって敵対的な攻撃から分散型学習を守ることは非常に重要になってきている.

分散型学習のパラメータ更新の際の集約規則をロバストにすることについて議論する. 最近では, よりロバストにするために, 集約規則で Averaging の計算を用いることに代わって Median ベースの計算多く用いられている.Median ベースの計算は敵対者が一定の割合まではロバストであることが利点である.

しかしながら, 大規模なデータ設定では Median の計算するコストが勾配を計算するコストを上回ってしまい, 実用的ではない. さらに,Median の収束の証明は convex のような強い過程を必要とする. 敵対者に対してロバストで, 大規模な一連のトレーニングアルゴリズム (ミニバッチ SGD,GD,coordinate descent など) に適用可能な分散型トレーニングフレームワークは未解決の問題である.

この論文では, 符号理論を使ってロバストネスを保証する.DRACO という敵対的かつ最悪の場合の計算エラーに対してロバストな分散トレーニングフレームワークを提示する.(Figure 1 を参照)

P を計算ノード数,B を勾配数,. 冗長率 r とする. 各ラウンドで,P 個の各計算ノードは $\frac{B}{P}$ 個の勾配を計算し, その合計値をパラメータサーバに送る.

DRACO では, 各計算ノードが $\frac{rB}{P}$ 個の勾配を処理し, それらの線形結合をパラメータサーバに送る.Median の計算は訓練時間の大部分を占めるかもしれない. 一方 DRACO では, 計算のほとんどが計算ノードによって実行される. これにより,DRACO では分散学習で早い収束が得られる.

勾配の和 p を受け取ると, パラメータサーバは「復号」機能を使用して, 敵対ノードの効果を除去し, 元の勾配の合計 B を再構成する.DRACO では $\frac{r-1}{2}$ までの敵対者を許容できる. 許容数 $\frac{r-1}{2}$ は少ないと感じるかもしれない. しかし, 一定数のノードしか悪意のある現実的な状況では DRACO は非常に高速である.

DRACO の符号化方式は [1] に示されている手法に基づいている. ただし,[1] とは異なり,DRACO は敵対的な環境に合わせて作られている.

2 preliminaries

以下の経験誤差を最小化する.

$$\min_w \frac{1}{n}\sum_{i=1}^{n} \ell(w; x_i) \qquad\qquad x_i \in \mathbb{R}^m,\ w \in \mathbb{R}^d$$

経験誤差の最小化のために SGD を用いる. 以下のようにパラメータ更新をする.$i_k \in \{1, \ldots, n\}$ はランダムなデータ点のインデックスである.

$$w_k = w_{k-1} - \gamma \nabla \ell(w_{k-1}; x_{i_k})$$

以下はミニバッチ SGD のパラメータ更新である.$S_k \subseteq \{1, \ldots, n\}$ はランダムなデータ点のインデックの部分集合である.

$$w_k = w_{k-1} - \frac{\gamma}{|S_k|}\sum_{i \in S_k} \nabla \ell(w_{k-1}; x_{i_k})$$

この論文では, 上記の更新を分散してロバストに実行する方法を検討する.S_k を $\{1, \ldots, B\}$ に置き換え,$\nabla \ell(w_{k-1}; x_{i_k})$ を g_k で表す. となると,$\sum_{i=1}^{B} g_i$ を敵対的分散設定で計算するのが目標である.

P 個の計算ノードのうちの部分集合が敵対的な挙動をすると仮定する. 敵対者はパラメータ更新を邪魔することで, 最終的なモデルを壊すことが目標である. 敵対者は完全な知識を持っていると仮定する.

3 DRACO: Robust Distributed Training via Algorithmic Redundancy

P は計算ノードで B はデータ点である.

$P \times B$ の行列 A を用いてデータ点 B の勾配を計算ノード P に割り当てる. もし計算ノード j が k 番目の勾配 g_k に割り当てられているのならば,$A_{j,k}$ は 1 となり, 割り当てられていない場合は,0 となる.

以降 $P = B$ と仮定する.

$d \times P$ の行列 $G \triangleq [g_1, g_2, \ldots, g_P]$ と定義する. また,j 番目の計算ノードは $d \times P$ 行列 $Y_j \triangleq (1_d A_j, .) \odot G$ に割り当てられている.

j 番目の計算ノードは, 割り当てられた勾配の $d \times p$ 行列 Y_j を d 次元ベクトルに写像する符号化関数 E_j がある.j 番目の計算ノードは割り当てられた勾配を計算した後 $z_j \triangleq E_j(Y_j)$ をパラメータサーバに送る.j 番目の計算ノードが敵対的である場合, 代わりに $z_j + n_j$ を送る.$E = \{E_1, E_2, \ldots, E_P\}$ とする.

$d \times P$ の行列 $Z^{A,E,G} \triangleq [z_1, z_2, \ldots, z_P]$ と定義する. また $d \times P$ の行列 $N \triangleq [n_1, n_2, \ldots, n_P]$ と定義する. 行列 N は敵対者の送るズレの値である. つまり, パラメータサーバは $d \times P$ の行列 $R \triangleq Z^{A,E,G} + N$ を受け取る. そして, 復号関数を D とすると, パラメータサーバは $u \triangleq D(R)$ を計算する.

DRACO は A, E, D によって決まる. A は勾配の割り当て方法,E は勾配を局所的にまとめる方法, さらに,D は復号の方法を担っている. 敵対的なノードがどんな値を送っても, 複合化後は敵対的ノードを完全に排除した勾配の合計値を送る.DRACO の手順は (Figure 2.) を参照.

Theorem 3.1. r を冗長率,s を敵対的ノード数とすると $r \leq 2s + 1$ を満たすとき,s 個の敵対的ノードに対して耐えることができる.

Theorem 3.1 より各計算ノードは少なくとも $2s + 1$ 個の d 次元ベクトルを符号化する. したがって, 符号化が線形の複雑さを有する場合, 各符号化は, 最悪の場合 $(2s + 1)d$ 回の演算を必要とする. デコーダが線形の複

雑さを有する場合,P 個の計算ノードからの d 次元の入力を使用する必要があるので, 最悪の場合に少なくとも $P \cdot d$ 個の演算を必要とする. よってオーダーは $\mathcal{O}(Pd)$ となる. これは $\mathcal{O}(P^2(d+\log P))$ の Median アプローチ (krum)[2] の計算コストよりもかなり低い.

以下で [1] に基づく 2 つの手法を紹介する.

Fractional Repetition Code

次のように動作する. まず,P 個の各計算ノードを $r = 2s+1$ 個のグループに分割する. 一つのグループ内の勾配の和が等しくなるように各ノード内の勾配の係数を調節する.$\hat{g}$ を反復ごとの正しい合計値とする. デーコーダーは各ノードからパラメータサーバに送られた合計値の多数決を取る. ノードの半分以下が敵対的な限り,$\hat{g}$ を送る保証がされる.

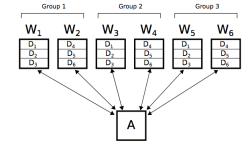

Figure 3: Fractional Repetition Scheme for $n = 6, s = 2$

Figure1　Fractional Repetition Code 概要

Theorem 3.2. 冗長率 $r = 2s+1$ とすると DRACO アルゴリズムは s 人の任意の敵に耐えられ, 最適な冗長率を達成し符号化および復号化は線形時間となる.

4 Experiments

DRACO-based mini-bathch SGD と mini-batch SGD と GM based mini-batch SGD[3] を比較する.

Algorithm 2 Byzantine Gradient Descent: Iteration $t \geq 1$

Parameter server:

1: *Initialize:* Let θ_0 be an arbitrary point in Θ; group the m machines into k batches, with the ℓ-th batch being $\{(\ell-1)b+1, \ldots, \ell b\}$ for $1 \leq \ell \leq k$.
2: Broadcast the current model parameter estimator θ_{t-1} to all working machines;
3: Wait to receive all the gradients reported by the m machines; If no message from machine j is received, set $\nabla \tilde{f}_j(\theta_{t-1})$ to be some arbitrary value;
4: *Robust Gradient Aggregation*

$$\mathcal{A}_k(\boldsymbol{g}_t(\theta_{t-1})) \leftarrow \mathsf{med}\left\{\frac{1}{b}\sum_{j=1}^{b} g_t^{(j)}(\theta_{t-1}), \cdots, \frac{1}{b}\sum_{j=m-b+1}^{n} g_t^{(j)}(\theta_{t-1})\right\}. \quad (8)$$

5: Update: $\theta_t \leftarrow \theta_{t-1} - \eta \times \mathcal{A}_k\big(\boldsymbol{g}_t(\theta_{t-1})\big)$;

Working machine j:

1: Compute the gradient $\nabla \tilde{f}^{(j)}(\theta_{t-1})$;
2: Send $\nabla \tilde{f}^{(j)}(\theta_{t-1})$ back to the parameter server;

Figure2　[3] で提案されているアルゴリズム

計算ノード数 45 とし, 敵対者数を $s = 1, 3, 5 (2.2\%, 6.7\%, 11.1\%)$ とする. それぞれ 10000 イテレーション訓練する. まず, 目的の accucy までの到達速度を検証する (Table 2,3 を参照). 逆向きの勾配を与える敵対者の下で,DRACO は GM より数倍高速に収束する. これは,GM の計算が DRACO の符号化と復号のオーバーヘッドよりも時間がかかるためであると考えられる.

収束性について検証する (Figure 3 を参照). 一定の敵対者の下では,DRACO はすべての実験で収束する

が,GM は収束しないことがあった.DRACO は常に, 敵対的な環境では通常のアルゴリズムで訓練されたモデルと全く同じモデルを返す.GM が収束しない理由の 1 つは, 幾何平均を用いることによって, 勾配についての情報を失ってしまうことである. もう一つの理由は,GM は損失関数が convex であることが収束の条件であることだからである.

次に 3 つの大規模なネットワーク (ResNet-152,VGG-19,AlexNet) に DRACO を適用して, イテレーション毎について検証する. 計算ノード数 45 とし, バッチサイズ $B = 40$ とする.$s = 5$ の場合,DRACO のエンコードとデコードの時間は通常の SGD の計算時間の数倍かかるが,SGD は敵対的設定では収束しないことがある. さらに,DRACO は GM より数倍高速である (Figure 4).

5 Conclusion and Open Problems

この論文では, 冗長性を利用したロバストな分散フレームワークである DRACO を紹介した.DRACO は、任意の敵対者に対してロバストであり, 高速である.2 つの今後の展望がある. まず,DRACO は敵対者の有無にかかわらず全く同じモデルを出力するように設計されている. しかし, わずかに異なるパラメータ更新でも, パフォーマンスが著しく落ちることはないということである. もう 1 つは, 比較的効率的なエンコードとデコードの方法を紹介したが, 分散型でより効率的な手法があるかもしれないということである.

References

[1] Tandon, Rashish, Lei, Qi, Dimakis, Alexandros G, and Karampatziakis, Nikos. Gradient coding: Avoiding stragglers in distributed learning.*Proceedings of the 35th International Conference on Machine Learning*, PMLR 3368 - 3376,2017.

[2] Blanchard, P., Mhamdi, E. M. E., Guerraoui, R., and Stainer, J. Machine Learning with Adversaries : Byzantine Tolerant Gradient Descent. In *Advances in Neural Information Processing Systems 30*,pages 119-129, 2017.

[3] Chen, Yudong, Su, Lili, and Xu, Jiaming. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. arXiv preprint arXiv:1705.05491, 2017.