

論文輪講

Federated Optimization:
Distributed Machine Learning for On-Device Intelligence

杉浦 圭祐

慶應義塾大学理工学部情報工学科 松谷研究室

May 11, 2019

- ① 問題設定
- ② 基本的な最適化アルゴリズム
- ③ ランダム化された最適化アルゴリズム
- ④ Federated Learning のための最適化アルゴリズム

目次

1 問題設定

- 問題の定式化

- 解くべき問題は次のように定式化される

$$\min_{\mathbf{w} \in \mathbb{R}^D} f(\mathbf{w}), \quad f(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{w}) \quad (1)$$

- 入出力のデータを $\{\mathbf{x}_i, y_i\}_{i=1}^N$ 、損失関数を $f_i(\mathbf{w})$ とする
- 具体的には、以下のような問題が考えられる

線形回帰：

$$f_i(\mathbf{w}) = \frac{1}{2} (\mathbf{x}_i^T \mathbf{w} - y_i)^2, \quad y_i \in \mathbb{R} \quad (2)$$

ロジスティック回帰：

$$f_i(\mathbf{w}) = -\log(1 + \exp(-y_i \mathbf{x}_i^T \mathbf{w})), \quad y_i \in \{-1, 1\} \quad (3)$$

サポートベクタマシン：

$$f_i(\mathbf{w}) = \max\{0, 1 - y_i \mathbf{x}_i^T \mathbf{w}\}, \quad y_i \in \{-1, 1\} \quad (4)$$

- 問題の定式化

- 上記は凸最適化問題である
- 線形回帰、ロジスティック回帰、サポートベクタマシンより複雑なモデルにも適用可能
 - ⇒ 条件付き確率場や、ニューラルネットワーク
 - ⇒ 損失関数 $f_i(\mathbf{w})$ が非凸で、複雑な形状をしている場合
- データ数 N が大き過ぎて、単一のノードで学習を進めるのは不可能
 - ⇒ 分散処理が必要 (データが複数の箇所に分散し、複数の相互接続されたノードで学習を行う)
 - ⇒ ノード間での通信時間がボトルネックとなる可能性
 - ⇒ 並列計算の利点を活かすために、モデルの学習を、単一ノードでの計算に適した簡単な問題に分割する必要がある

- 最新 (State-of-the-art) の最適化手法
 - シーケンシャルであるため、並列処理には向かない
 - 各イテレーションでの処理は非常に高速だが、イテレーションの実行を何度も繰り返す必要がある
 - ⇒ 各イテレーションの実行後に、複数のノード間で通信を行うと、性能が極端に落ちる
 - ⇒ 各イテレーションの実行時間より、ノード間の通信時間の方が遥かに大きい

- 従来手法における仮定

- 1 データは各ノードに均等に分散している

- 2 ノード数 K に比べて、1つのノードが保持しているデータ数の平均 N/K の方が非常に大きい ($K \ll N/K$)
⇒ 大規模なデータセンタに、データが格納されている想定

- 3 各ノードが、分布をよく表現するデータを保持している
⇒ 各ノードが、**独立同分布** (IID) 標本を持っているという前提

- ⇒ 実際には、ノードの地理的な位置によって、各ノードが持つデータが、クラスタに分かれている可能性

- ⇒ 各ノードが持つデータが時間的に変動し、ある時点では他のノードと似たようなデータを保持している可能性

- ⇒ あるノードに頻繁に現れる特徴が、他のノードでは全く現れない可能性

- 従来手法における仮定

- 従来手法では、上記 3 つの仮定が成立する
 - ⇒ Federated Learning では、これらの**仮定を全く置かない**
- 従来手法では、最初に、デバイス上のデータを中央のノード (データセンタ等) に集める
 - ⇒ 集めたデータをランダムにシャッフルし、複数の計算ノードに均等な数だけ配分して、学習させる
- Federated Learning では、デバイス上のデータを中央のノードに送信しない
 - ⇒ 各デバイスと中央のノードとの通信量が大幅に削減
 - ⇒ ユーザのプライバシーを保護し、セキュリティを向上させる
 - ⇒ データがデバイス上にしか存在しないので、中央のノードが攻撃されても、ユーザのデータが漏洩する危険性がない (攻撃されそうな箇所の候補が減る)

- Federated Learning での問題設定

- Federated Learning では、次のような現実的な仮定を置く

- Massively Distributed**: データは、多数のノードに分散
⇒ ノード数 K と、1つのノードが保持しているデータ数の平均 N/K を比較したとき、 $N/K \ll K$ となる可能性
- Non-IID**: 各ノードが保持するデータは、異なる分布からサンプルされた可能性
⇒ あるノードが保持しているデータは、データ全体の分布を表現しているとは限らない
⇒ 各ノードが保持するデータは、独立に同一の分布からサンプルされた (IID) とは、仮定し難い
- Unbalanced**: ノードによって、保持しているデータ数は大きく異なる

- Federated Learning での問題設定

- 上記に加えて、この論文では次のような仮定を置く

- 1 データはスパースである

⇒ ある特徴は、ほんの僅かなノードやデータにしか現れない

- 2 データはモバイル端末上に存在し、プライバシー上の配慮が必要 (Privacy Sensitive)

⇒ 入出力データ $\{x_i, y_i\}$ はデバイス上で作られる

⇒ 例えば、ユーザが次に入力する単語の予測、ユーザがシェアしそうな写真の予測、ユーザにとってどの通知が重要かの予測

- 3 多数のデバイスが動作するので、事実上無限の計算能力が得られる
 - ⇒ 但し、各デバイスと中央のノード間の、通信コストによって制限される (バンド幅の制限)
 - ⇒ デバイスと中央のノード間の通信を、いかに減らせるかが、性能向上のための鍵となる

- 4 差分データ $\delta \in \mathbb{R}^D$ が、モデルの学習に使用する**唯一の情報**である
 - ⇒ 各デバイスは、1 回の Roundにつき、モデルのパラメータの差分 $\delta \in \mathbb{R}^D$ を計算 (D は、モデルのパラメータの次元)
 - ⇒ 差分 δ が、デバイスから中央のノードにアップロードされる

- Federated Learning で送信される差分データ δ について
 - δ は、ユーザのプライベートな情報を依然として含むかもしれないが、訓練データ $\{x_i, y_i\}$ に比べれば無視できるほど小さい
 - モデルのパラメータの差分ベクトル δ のサイズは、訓練データのサイズには関係ない
⇒ 訓練データが巨大 (例えばユーザが撮影した動画) であっても、データそのものではなく、差分データのみを中央のノードに送信すればよい
ため、**通信量が大幅に削減**できる

- 差分ベクトル δ の使途は、元の訓練データ $\{x_i, y_i\}$ に比べて限られる
 - ⇒ モデルの訓練以外に、殆ど使い道がない
 - ⇒ 各デバイスから送信された差分データは、モデルの訓練に使用した後、破棄してよい
- ⇒ ユーザにとっては、アップロードしたデータが、モデルの訓練にしか使われない (想定外の方法で使われない) ことが分かっているので、安心できる (**プライバシーの保護**につながる)
- ⇒ モデルを訓練する側にとっては、ユーザのデータを保存する際の負担が軽減される
- ⇒ 訓練データであれば、漏洩しないように厳重に管理する必要がある
- ⇔ 差分データであれば、万が一不正にアクセスされても、ユーザの個人情報漏洩することはない

- 論文で提案された訓練アルゴリズムについて
 - Federated Learning のための訓練アルゴリズムを新たに設計した
⇐ **Massively Distributed**、**Non-IID**、**Unbalanced**
 - 新たな訓練アルゴリズムによって、比較的少ない Round 数 (通信量) で、モデルのパラメータを収束させることができた

② 基本的な最適化アルゴリズム

基本的な最適化アルゴリズム

- 解くべき問題は、次のように定式化された
 - D はモデルのパラメータの次元数、 N は訓練データ数
 - $\mathbf{w} \in \mathbb{R}^D$ はパラメータベクトル、 $f_i(\mathbf{w})$ は損失関数

$$\min_{\mathbf{w} \in \mathbb{R}^D} f(\mathbf{w}), \quad f(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{w}) \quad (5)$$

- ベースラインアルゴリズム
 - 基本的なアルゴリズムとして、以下を紹介する
 - 勾配降下法 (GD; Gradient Descent)
 - 確率的勾配降下法 (SGD; Stochastic Gradient Descent)

基本的な最適化アルゴリズム

- 勾配降下法 (GD; Gradient Descent)
 - パラメータの更新式は次のようになる

$$\mathbf{w}^{t+1} = \mathbf{w}^t - h_t \nabla f(\mathbf{w}^t) \quad (6)$$

- $\nabla f(\mathbf{w}^t)$ は次のように定義される

$$\nabla f(\mathbf{w}^t) \equiv \left. \frac{\partial}{\partial \mathbf{w}} f(\mathbf{w}) \right|_{\mathbf{w}=\mathbf{w}^t} \quad (7)$$

$$= \frac{1}{N} \sum_{i=1}^N \left. \frac{\partial}{\partial \mathbf{w}} f_i(\mathbf{w}) \right|_{\mathbf{w}=\mathbf{w}^t} \quad (8)$$

$$= \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{w}^t) \quad (9)$$

損失関数 $f(\mathbf{w})$ のパラメータ \mathbf{w} による勾配の、 $\mathbf{w} = \mathbf{w}^t$ における値

- $h_t > 0$ は学習率 (ステップサイズ)

基本的な最適化アルゴリズム

- 勾配降下法 (GD; Gradient Descent) の問題点
 - 勾配 $\nabla f(\mathbf{w}^t)$ を求めるためには、 N 個の各データについての勾配 $\nabla f_i(\mathbf{w}^t)$ を計算する必要がある
 - ⇒ 1 回のパラメータ更新のために、全データを処理する必要がある
 - ⇒ データ数 N は非常に大きいため、勾配の計算に時間が掛かり過ぎる
 - ⇒ 勾配降下法は、遅すぎて使い物にならない
- モメンタム項を加えることで、アルゴリズムを高速化できる
 - ⇒ 但し、1 回のパラメータ更新のために、全データを処理する必要がある
 - ⇒ モメンタム項を加えても、やはり使い物にならない

基本的な最適化アルゴリズム

- 確率的勾配降下法 (SGD; Stochastic Gradient Descent)

- パラメータの更新式は次のようになる

$$\mathbf{w}^{t+1} = \mathbf{w}^t - h_t \nabla f_{i_t}(\mathbf{w}^t) \quad (10)$$

- $\nabla f_{i_t}(\mathbf{w}^t)$ は次のように定義される

$$\nabla f_{i_t}(\mathbf{w}^t) = \left. \frac{\partial}{\partial \mathbf{w}} f_{i_t}(\mathbf{w}) \right|_{\mathbf{w}=\mathbf{w}^t} \quad (11)$$

- i_t は、1 から N の中から適当に選択されたインデックス
⇒ 時刻 t では、 i_t 番目のデータ $\{\mathbf{x}_{i_t}, y_{i_t}\}$ とパラメータ \mathbf{w}^t を用いて、損失関数 $f_{i_t}(\mathbf{w}^t)$ を計算
⇒ 損失関数の勾配 $\nabla f_{i_t}(\mathbf{w}^t)$ を求めて、パラメータ \mathbf{w} を勾配の方向に更新

基本的な最適化アルゴリズム

- 確率的勾配降下法 (SGD; Stochastic Gradient Descent)

- 1 回のパラメータ更新のためには、1 つのデータ点に対する勾配 $\nabla f_{i_t}(\mathbf{w}^t)$ だけを求めればよい
⇒ 勾配降下法とは異なり、全データを処理する必要がない

- 1 つのデータ点に対する勾配の期待値は、損失関数 $f(\mathbf{w})$ の勾配の不変推定量となっている
⇒ $\mathbb{E}[\nabla f_{i_t}(\mathbf{w})] = \nabla f(\mathbf{w}^t)$ であり、この手法が正当化される

⇒ 実際には、データ点のサンプリングによって、(真の勾配に対して) ノイズが加わった勾配が得られるため、パラメータの収束が遅くなる
⇒ 学習率 (ステップサイズ) h_t の設定が重要になる

- 勾配の計算に用いるデータ点 i_t を、毎回ランダムに選ぶのではなく、全データをランダムな順で取り出し、その勾配を求めてパラメータを更新していく手法がある

基本的な最適化アルゴリズム

- GD と SGD との比較

- GD ではパラメータの収束が速い \Leftrightarrow SGD は収束が遅い
- GD では各イテレーションの計算に非常に時間が掛かる
 - \Leftarrow 各イテレーションにおいて、全データを処理する必要がある
 - \Leftrightarrow SGD では各イテレーションの計算は高速であり、計算時間はデータ数 N に依存しない
- 今回解こうとしている問題では、パラメータの精度はそこまで高くなくてもよい
 - \Rightarrow SGD で十分である (極端な場合には、全データを 1 回ずつ処理するだけで、パラメータが収束)
 - \Leftarrow GD の場合は、全データを 1 回ずつ処理して、ようやくパラメータを 1 回更新できる

③ ランダム化された最適化アルゴリズム

ランダム化された最適化アルゴリズム

- ランダム化された座標降下法 (RCD; Randomized Coordinate Descent)
 - パラメータの更新式は次のようになる

$$\mathbf{w}^{t+1} = \mathbf{w}^t - h_{j_t} \nabla_{j_t} f(\mathbf{w}^t) \mathbf{e}_{j_t} \quad (12)$$

- j_t は、1 から D (パラメータの次元数) の中から適当に選択された次元
- h_{j_t} は学習率 (ステップサイズ)、 $\mathbf{e}_{j_t} \in \mathbb{R}^D$ は次元 j_t 方向の標準基底ベクトル
- 勾配 $\nabla_j f(\mathbf{w}^t)$ は次のように定義される

$$\nabla_j f(\mathbf{w}^t) = \frac{\partial}{\partial w_j} f(\mathbf{w}^t) \quad (13)$$

$$= \frac{1}{N} \sum_{i=1}^N \left. \frac{\partial}{\partial w_j} f_i(\mathbf{w}) \right|_{\mathbf{w}=\mathbf{w}^t} \quad (14)$$

$$= \frac{1}{N} \sum_{i=1}^N \nabla_{w_j} f_i(\mathbf{w}^t) \quad (15)$$

ランダム化された最適化アルゴリズム

- ランダム化された座標降下法 (RCD; Randomized Coordinate Descent)
 - 最適化問題 $\min_{\mathbf{w} \in \mathbb{R}^D} f(\mathbf{w})$ を、幾つかの部分問題に分割
 - $f(\mathbf{w})$ を、 $j \neq j_t$ であるような変数については固定したまま、ある 1 つの変数 $j_t \in \{1, \dots, D\}$ について最小化する
⇒ 一度に 1 つの座標を最適化するため、**座標降下法**とよばれる
 - ⇒ 一般に、変数の部分集合について同時に最小化を行うアルゴリズムを、**ブロック座標降下法**という
- 座標降下法は、ある 1 つの変数が、他の変数の最適値に影響を与えない場合に効果を発揮
⇒ 「深層学習」の 8.7.2 節を参照

ランダム化された最適化アルゴリズム

- SDCA; Stochastic Dual Coordinate Ascent

- 正則化項 $\psi(\mathbf{w})$ を付加した、以下の最適化問題を考える

$$\min_{\mathbf{w} \in \mathbb{R}^D} f(\mathbf{X}\mathbf{w}) + \psi(\mathbf{w}), \quad f(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N f_i(y_i, \mathbf{x}_i^T \mathbf{w}) \quad (16)$$

- 上記の**双対問題** (Dual problem) は、**Fenchel 双対定理**から、次のようになる (K はデータ \mathbf{x} の次元数) [2]

$$- \min_{\mathbf{u} \in \mathbb{R}^N} f^*(\mathbf{u}) + \psi^* \left(-\frac{\mathbf{X}^T \mathbf{u}}{N} \right) \quad (17)$$

- 但し、 f^* と ψ^* は、それぞれ f と ψ の**ルジャンドル変換**である

ランダム化された最適化アルゴリズム

- SDCA; Stochastic Dual Coordinate Ascent

- $f(\mathbf{x})$ のルジャンドル変換 $f^*(\mathbf{y})$ とは、関数 $f(\mathbf{x})$ の変数を、その微分 $\mathbf{y} = \mathbf{x}'$ に置き換えた関数であり、次のように定義される (関数 $f(\mathbf{x})$ を、その傾きの情報から捉えた関数)

$$f^*(\mathbf{y}) = \sup_{\mathbf{x}} \{ \mathbf{x}^T \mathbf{y} - f(\mathbf{x}) \} \quad (18)$$

- 双対問題 (Dual problem) とは、最適化問題における主問題 (Primary problem) の補問題であり、主問題と双対問題は表裏一体の関係にある

ランダム化された最適化アルゴリズム

- SDCA; Stochastic Dual Coordinate Ascent

- 例えば、以下の問題 P (主問題) を、次のように定める

$$\min_x c^T x \quad \text{s.t.} \quad Ax = b, x \geq 0 \quad (19)$$

- 一方の問題 D (双対問題) は、次のようになる

$$\max_y b^T y \quad \text{s.t.} \quad A^T y \leq c \quad (20)$$

- 双対定理より、問題 P に最適解 x^* が存在すれば、問題 D にも最適解 y^* が存在して、 $c^T x^* = b^T y^*$ が成立

ランダム化された最適化アルゴリズム

- SDCA; Stochastic Dual Coordinate Ascent

- 今回の場合、問題 P (主問題) は、(正則化項付きの) 損失関数の最小化である
- 問題 P の代わりに双対問題 D を解くことができる
- このとき、問題 P の解が得られるので、最適なパラメータが求まる
- SDCA は、このような考え方に基づくアルゴリズムである (詳細は省略)
- 主問題と双対問題を、交互に解くアルゴリズムも考えられる
- 以降のスライドでは、勾配 $\nabla f(w)$ の推定値に含まれるノイズを軽減するアルゴリズムをみていく

分散を抑えた確率的勾配降下法

- SAG; Stochastic Average Gradient

- 確率的勾配の平均を取るアルゴリズム
- SAG は、GD(勾配降下法) と SGD(確率的勾配降下法) の中間に位置
- SAG の各イテレーションでは、 $i_t \in \{1, \dots, N\}$ を選択したうえで、次の処理が実行される

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \frac{\alpha_t}{N} \sum_{i=1}^N \mathbf{y}_i^t \quad (21)$$

$$= \mathbf{w}^t - \frac{\alpha_t}{N} \left[\sum_{i=1}^N \mathbf{y}_i^{t-1} + (\mathbf{y}_{i_t}^t - \mathbf{y}_{i_t}^{t-1}) \right] \quad (22)$$

$$= \mathbf{w}^t - \frac{\alpha_t}{N} \left[\sum_{i=1}^N \mathbf{y}_i^{t-1} + (\nabla f_{i_t}(\mathbf{w}^t) - \mathbf{y}_{i_t}^{t-1}) \right] \quad (23)$$

分散を抑えた確率的勾配降下法

- 但し、 \mathbf{y}_i^t は次のように定義される

$$\mathbf{y}_i^t = \begin{cases} \nabla f_i(\mathbf{w}^t) & (i = i_t) \\ \mathbf{y}_i^{t-1} & (\text{それ以外のとき}) \end{cases} \quad (24)$$

- 勾配 $\nabla f_i(\mathbf{w}^t)$ は次のように定義される

$$\nabla f_i(\mathbf{w}^t) \equiv \left. \frac{\partial}{\partial \mathbf{w}} f_i(\mathbf{w}) \right|_{\mathbf{w}=\mathbf{w}^t} \quad (25)$$

分散を抑えた確率的勾配降下法

- SAG; Stochastic Average Gradient

- GD では、現在のパラメータ w^t を基に、**全てのデータについて** 勾配 $\nabla f_i(w^t)$ を計算
 - ⇒ その平均 $\sum_{i=1}^N \nabla f_i(w^t)$ (**Full gradient**) を使って、パラメータ w を更新
 - ⇒ 1 回のパラメータの更新には Full gradient が必要であり、全てのデータを使って計算するため、非常に処理が重い
- SGD では、データ点 $i \in \{1, \dots, N\}$ についてのみ (**1 つのデータについてのみ**)、勾配 $\nabla f_i(w^t)$ を計算
 - ⇒ その勾配 $\nabla f_i(w^t)$ を使って、パラメータ w を更新
 - ⇒ 1 点における勾配 $\nabla f_i(w^t)$ を用いて、 $\sum_{i=1}^N \nabla f_i(w^t)$ を近似
 - ⇒ 1 回のパラメータの更新に必要な、計算量を少なく抑えられる

分散を抑えた確率的勾配降下法

- SAG; Stochastic Average Gradient

- SAG では、全データに対する勾配 (Full gradient) を、**少しずつ更新していく**

⇒ ある 1 つのデータ点 $i_t \in \{1, \dots, N\}$ について、現在のパラメータ \mathbf{w}^t の下で勾配 $\nabla f_{i_t}(\mathbf{w}^t)$ を計算

⇒ 新しく求めた勾配 $\nabla f_{i_t}(\mathbf{w}^t)$ を使って、以下の式で Full gradient を更新

$$\sum_{i=1}^N \mathbf{y}_i^t = \sum_{j \neq i_t} \mathbf{y}_j^{t-1} + \nabla f_{i_t}(\mathbf{w}^t) \quad (26)$$

⇒ 更新された Full gradient を使って、パラメータ \mathbf{w} を更新

- SAG では、計算される勾配にはバイアスが含まれる
⇒ 後述の **SAGA** で計算される勾配の期待値は、勾配 $f(\mathbf{w})$ の不変推定量に一致

分散を抑えた確率的勾配降下法

- SAG; Stochastic Average Gradient

- SAG では、各データにおける勾配の履歴 (現在の Full gradient)、従って $\{\mathbf{y}_i^t\}_{i=1}^N$ を記憶しなければならないことが分かる
- 比較的小規模のニューラルネットの学習であっても、勾配を記憶するためのメモリ使用量が大きいため、アルゴリズムは使い物にならなくなる
- GD の速い収束と、SGD の速い計算時間という、双方の利点を受け継いだアルゴリズム
- SAG の改良版として、SAGA アルゴリズムが存在 (詳細は省略)
- 因みに、SAGA が何の略称なのかは不明

分散を抑えた確率的勾配降下法

- SAGA におけるパラメータの更新式は次のようになる [1]

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \alpha_t \left[\frac{1}{N} \sum_{i=1}^N \mathbf{y}_i^{t-1} + (\mathbf{y}_{i_t}^t - \mathbf{y}_{i_t}^{t-1}) \right] \quad (27)$$

$$= \mathbf{w}^t - \alpha_t \left[\frac{1}{N} \sum_{i=1}^N \mathbf{y}_i^{t-1} + (\nabla f_{i_t}(\mathbf{w}^t) - \mathbf{y}_{i_t}^{t-1}) \right] \quad (28)$$

分散を抑えた確率的勾配降下法

- SVRG; Stochastic Variance Reduced Gradient

- 二重ループの最適化アルゴリズムである
- 外側のループでは、Full gradient(全データについての勾配の平均) $\nabla f(\mathbf{w}^t)$ を計算

$$\nabla f(\mathbf{w}^t) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{w}^t) = \frac{1}{N} \sum_{i=1}^N \left. \frac{\partial}{\partial \mathbf{w}} f_i(\mathbf{w}) \right|_{\mathbf{w}=\mathbf{w}^t} \quad (29)$$

- 内側のループでは、インデックス $i \in \{1, \dots, N\}$ を選択し、次の式を用いてパラメータを更新していく

$$\mathbf{w} = \mathbf{w} - h [\nabla f_i(\mathbf{w}) - \nabla f_i(\mathbf{w}^t) + \nabla f(\mathbf{w}^t)] \quad (30)$$

- 確率的勾配 $\nabla f_i(\mathbf{w}) - \nabla f_i(\mathbf{w}^t)$ は、 \mathbf{w} と \mathbf{w}^t における勾配の変化 $\nabla f(\mathbf{w}) - \nabla f(\mathbf{w}^t)$ を推定するための項

分散を抑えた確率的勾配降下法

- SVRG が完全な勾配 $\nabla f(\mathbf{w}^t)$ を計算している間、パラメータは一度も更新されないが、同じ時間で、SGD ではパラメータが N 回更新される
⇒ 最初は、SGD の方が学習が進むことが予測される

分散を抑えた確率的勾配降下法

Algorithm 1: SVRG; Stochastic Variance Reduced Gradient [1]

parameters: m = number of stochastic steps per epoch, h = stepsize
(learning rate)

- 1: **for** $s = 0, 1, \dots$ **do**
 - 2: Compute and store full gradient $\nabla f(\mathbf{w}^t) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{w}^t)$
 // Full pass through data
 - 3: Set $\mathbf{w} = \mathbf{w}^t$
 - 4: **for** $t = 1$ **to** m **do**
 - 5: Pick $i \in \{1, \dots, N\}$, uniformly at random
 - 6: Update using $\mathbf{w} = \mathbf{w} - h [\nabla f_i(\mathbf{w}) - \nabla f_i(\mathbf{w}^t) + \nabla f(\mathbf{w}^t)]$
 // Stochastic update
 - 7: **end for**
 - 8: $\mathbf{w}^{t+1} = \mathbf{w}$
 - 9: **end for**
-

分散を抑えた確率的勾配降下法

- アルゴリズムについての補足

- SGD と比較すると SVRG の性能は良く、各イテレーションでの分散が小さい
- SVRG と、ランダム化された座標降下法 (RCD; Randomized Coordinate Descent) とを結びつけるアルゴリズムが存在
- SAGA が提唱された論文では、SAGA は SAG と SVRG の中間に位置付けられている
- SVRG、SAGA、SAG、GD を一般化したアルゴリズムが登場している

4 Federated Learning のための最適化アルゴリズム

Federated Learning のための最適化アルゴリズム

- Federated Learning のための最適化アルゴリズム
 - **SVRG**(Stochastic Variance Reduced Gradient) アルゴリズムと、**DANE**(Distributed Approximate Newton) アルゴリズムを調べる
 - SVRG と DANE は一見無関係に見えるが、実は深く関連し合う
 - SVRG を改良した **Federated SVRG** について説明する
 - Federated SVRG を更に改良したアルゴリズムを、新たに提案する
⇐ 各デバイスに保存された訓練データの個数、訓練データのスパース性、各デバイス上の訓練データのパターンの相違について考慮
 - Federated Learning では、訓練データが **Massively Distributed**、**Non-IID**、**Unbalanced** であるという仮定を置く
 - これに加え、**スパース性**、**Privacy Sensitive** などの仮定を設けた

Federated Learning のための最適化アルゴリズム

- Federated Learning のためのアルゴリズムに必要な特徴

- 1 アルゴリズムの開始時に、パラメータが既に最適値であったなら、そのアルゴリズムを何度実行しても、パラメータの値が変化しない
- 2 訓練データが単一のノードにしかないとき、パラメータが収束するまでに必要な、中央のノードとの通信回数は $\mathcal{O}(1)$ に抑えられること
- 3 データの各特徴が、単一のノードにしか現れないとき (解こうとしている問題が完全に分離され、各デバイスがパラメータの一部を学習しているとき)、 $\mathcal{O}(1)$ 回の通信回数の後に、パラメータが収束すること

⇐ データの各次元が、ある 1 つのノードでは 1 になるが、他の全てのノードでは 0 になるような場合
- 4 各ノードが完全に同一なデータセットを有するとき、 $\mathcal{O}(1)$ 回の通信回数の後に、パラメータが収束すること

Federated Learning のための最適化アルゴリズム

- Federated Learning のためのアルゴリズムに必要な特徴

- 「収束する」とは、「十分に精度のある適当な解が得られる」ことを意味している

⇒ $O(1)$ 回とは、各デバイスと中央のノード間で、**たった 1 度だけ**やり取りすることに相当

- (1) は、全ての最適化問題において、考慮する価値がある設定
- (2) と (3) は、Federated Learning における極端なケース
- (4) は、従来の最適化問題における設定 (中央の少数のノードが多量のデータを保持している状況)

⇒ (4) は、Federated Learning においては最も重要でない

Algorithm 2: SVRG; Stochastic Variance Reduced Gradient (Recall) [1]

parameters: m = number of stochastic steps per epoch, h = stepsize
(learning rate)

- 1: **for** $s = 0, 1, \dots$ **do**
 - 2: Compute and store full gradient $\nabla f(\mathbf{w}^t) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{w}^t)$
 // Full pass through data
 - 3: Set $\mathbf{w} = \mathbf{w}^t$
 - 4: **for** $t = 1$ **to** m **do**
 - 5: Pick $i \in \{1, \dots, N\}$, uniformly at random
 - 6: Update using $\mathbf{w} = \mathbf{w} - h [\nabla f_i(\mathbf{w}) - \nabla f_i(\mathbf{w}^t) + \nabla f(\mathbf{w}^t)]$
 // Stochastic update
 - 7: **end for**
 - 8: $\mathbf{w}^{t+1} = \mathbf{w}$
 - 9: **end for**
-

- SVRG; Stochastic Variance Reduced Gradient

- 外側のループでは、Full gradient(全データについての勾配の平均) $\nabla f(\mathbf{w}^t)$ を計算
- 内側のループでは、確率的勾配によるパラメータの更新を m 回実行
 - ⇒ m は、データ数 N の 1 倍から 5 倍程度の値に設定
 - ⇒ m は、実用上は N にすることが多い
- 内側のループでは、データ点 i における勾配 $\nabla f_i(\mathbf{w}), \nabla f_i(\mathbf{w}^t)$ を計算
 - ⇒ これらの勾配の差 $\nabla f_i(\mathbf{w}) - \nabla f_i(\mathbf{w}^t)$ を求めて、 \mathbf{w} と \mathbf{w}^t における Full gradient の差分 $\nabla f(\mathbf{w}) - \nabla f(\mathbf{w}^t)$ を推定するための項とする
 - ⇒ $\nabla f_i(\mathbf{w}) - \nabla f_i(\mathbf{w}^t) + \nabla f(\mathbf{w}^t)$ は、 $\nabla f(\mathbf{w})$ の **不変推定量** を導く

- SVRG; Stochastic Variance Reduced Gradient

- 更新中の \mathbf{w} と、固定された \mathbf{w}^t との差が小さければ、
 $\nabla f_i(\mathbf{w}) - \nabla f_i(\mathbf{w}^t)$ も小さくなるので、 $\nabla f(\mathbf{w})$ (の予測値) に加わるノイズも小さくなっていると予想できる
- 内側のループを実行する度に、 \mathbf{w} が \mathbf{w}^t から離れていくので、
 $\nabla f_i(\mathbf{w}) - \nabla f_i(\mathbf{w}^t)$ が増大し、従って Full gradient $\nabla f(\mathbf{w}^t)$ に加わるノイズが増大
⇒ 外側のループが実行され、新たな Full gradient $\nabla f(\mathbf{w}^{t+1})$ が計算されると、Full gradient に加わるノイズは再び小さくなる
- 関数 $f = \frac{1}{N} \sum_i f_i$ が λ -strongly convex function で、各 f_i が L -smooth function ならば、収束度合いは次のように表される (詳細は論文を参照)

$$\mathbb{E} [f(\mathbf{w}^t) - f(\mathbf{w}^*)] \leq c^t [f(\mathbf{w}^0) - f(\mathbf{w}^*)] \quad (31)$$

\mathbf{w}^* は $f(\mathbf{w})$ を最小化する最適解、 $c = \Theta\left(\frac{1}{mh}\right) + \Theta(h)$ (詳細は略)

- Federated Learning のための問題設定

- 解くべき問題は、経験損失 $f(\mathbf{w})$ の最小化であった

$$\min_{\mathbf{w} \in \mathbb{R}^D} f(\mathbf{w}), \quad f(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{w}) \quad (32)$$

- 各 f_i は凸関数で、訓練データ $\{\mathbf{x}_i, y_i\}_{i=1}^N$ が与えられている (多数のノードに不均等に分散)

- 分散学習のために、以下の記法を導入する

- K を **ノード数** とする
- \mathcal{P}_k を、ノード $k \in \{1, \dots, K\}$ が持つ訓練データの **インデックス集合** とする
- ノード k が持つ訓練データの個数を、 $N_k = |\mathcal{P}_k|$ と表す
 $\Rightarrow k \neq l$ のとき $\mathcal{P}_k \cap \mathcal{P}_l = \emptyset$ (空集合)、そして $\sum_{k=1}^K N_k = N$

- Federated Learning のための問題設定

- 経験損失 $f(\mathbf{w})$ を、次の手順で書き直す

$$F_k(\mathbf{w}) \equiv \frac{1}{N_k} \sum_{i \in \mathcal{P}_k} f_i(\mathbf{w}) \quad (33)$$

$$\begin{aligned} f(\mathbf{w}) &= \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^K \sum_{i \in \mathcal{P}_k} f_i(\mathbf{w}) \\ &= \frac{1}{N} \sum_{k=1}^K N_k \cdot \frac{1}{N_k} \sum_{i \in \mathcal{P}_k} f_i(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^K N_k F_k(\mathbf{w}) \end{aligned} \quad (34)$$

- $F_k(\mathbf{w})$ は、各ノード k が最小化すべき目的関数である (凸関数)
- これより、解くべき問題は次のように書き直される

$$\min_{\mathbf{w} \in \mathbb{R}^D} f(\mathbf{w}), \quad f(\mathbf{w}) = \sum_{k=1}^K \frac{N_k}{N} F_k(\mathbf{w}) \quad (35)$$

- Federated Learning のための問題設定

- この問題を解くための最も簡単な手法は、次の通りである

$$\mathbf{w}_k^{t+1} = \arg \min_{\mathbf{w} \in \mathbb{R}^D} F_k(\mathbf{w}), \quad \mathbf{w}^{t+1} = \sum_{k=1}^K \frac{N_k}{N} \mathbf{w}_k^{t+1} \quad (36)$$

- 各ノード上で目的関数 F_k を最小化し、得られた解 \mathbf{w}_k^{t+1} の N_k による重み付け和を \mathbf{w} とする
- この場合、上の問題を一度だけ解けば、解 \mathbf{w} が得られる (\mathbf{w}_k^{t+1} の右辺は t には依存しない) ので、各デバイスと中央のノードとの一度だけのやりとりで済む

- Federated Learning のための問題設定

- 上記のアルゴリズムは動作しない

- ⇐ 全体の解 w が、個々の解 w_k の重み付け和になっているとは考えにくい

- ⇐ 関数 F_k の形が、全ての k について等しいならば、重み付け和にはなっている

- ⇐ 関数の形が全て等しいならば、単一のノード上で $\min_{w \in \mathbb{R}^D} F_1(w)$ を解けばよいので、分散アルゴリズムを考える必要はない

- 分散アルゴリズムを導出したいが、上記のアルゴリズムでは無意味
 - 但し、各ノード k が、目的関数 F_k に含まれる曲がり具合の情報 (Curvature information) を最大限活用できるようにしたい

- Federated Learning のための問題設定

- 分散アルゴリズムを導出するために、各 F_k に二次の項 $-(\mathbf{a}_k^t)^T \mathbf{w} + \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}^t\|^2$ を摂動として加算する
- そして、各ノードが次の問題を解くようにする

$$\mathbf{w}_k^{t+1} = \arg \max_{\mathbf{w} \in \mathbb{R}^D} \left(F_k(\mathbf{w}) - (\mathbf{a}_k^t)^T \mathbf{w} + \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}^t\|^2 \right) \quad (37)$$

$$\mathbf{w}^{t+1} = \frac{1}{K} \sum_{k=1}^K \mathbf{w}_k^{t+1} \quad (38)$$

- 各ノード k が、目的関数 F_k に含まれる曲がり具合の情報 (Curvature information) を最大限活用できるようにしたい
 \Rightarrow 各ノードが最適化する関数の、ヘッセ行列は $\nabla^2 F_k + \mu \mathbf{I}$ となるので、関数 F_k に含まれる勾配の情報は、殆どそのまま保存される

- Federated Learning のための問題設定

- 以下の式を解きたいが、ベクトル \mathbf{a}_k^t の決め方が分からない

$$\mathbf{w}_k^{t+1} = \arg \max_{\mathbf{w} \in \mathbb{R}^D} \left(F_k(\mathbf{w}) - (\mathbf{a}_k^t)^T \mathbf{w} + \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}^t\|^2 \right)$$

- $t \rightarrow \infty$ の極限では、 \mathbf{w} が最適 ($\mathbf{w} = \mathbf{w}^*$) であるとき、上式の勾配が 0 となって欲しい

$$\begin{aligned} & \nabla \left(F_k(\mathbf{w}) - (\mathbf{a}_k^t)^T \mathbf{w} + \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}^t\|^2 \right) \\ &= \nabla F_k(\mathbf{w}) - \mathbf{a}_k^t + \mu (\mathbf{w} - \mathbf{w}^t) = 0 \end{aligned}$$

- 即ち、 $t \rightarrow \infty$ の極限では、 \mathbf{a}_k^t は次のようになって欲しい

$$\mathbf{a}_k^t = \nabla F_k(\mathbf{w}) + \mu (\mathbf{w} - \mathbf{w}^t) \simeq \nabla F_k(\mathbf{w}^*) \quad (\because \mathbf{w}^* \simeq \mathbf{w}^t)$$

- 但し、 \mathbf{w}^* を知らないなので、 $\mathbf{a}_k^t = \nabla F_k(\mathbf{w}^*)$ とはできない
 $\Rightarrow t \rightarrow \infty$ で、 $\mathbf{a}_k^t \rightarrow \nabla F_k(\mathbf{w}^*)$ となるような更新式を編み出す

- DANE; Distributed Approximate Newton
 - 先程の最適化問題は、双対問題と深く関連している
 - ⇒ 但し、上記のような問題がノード数分だけ存在するので、複雑である
 - DANE アルゴリズムでは、個々のノード上で解くための部分問題を構成することに主眼を置く
 - ⇐ 部分問題は、各ノード上のデータと、完全な勾配 $\nabla f(\mathbf{w}^t)$ にのみ依存
 - ⇐ Full gradient $\nabla f(\mathbf{w}^t)$ は、各デバイスと中央のノード間で、1 度だけやり取りすれば計算可能
 - DANE アルゴリズムを次に示す

Algorithm 3: DANE; Distributed Approximate Newton [?]

input : regularizer $\mu \geq 0$, parameter η (default: $\mu = 0, \eta = 1$)

- 1: Initialize \mathbf{w}^0
- 2: **for** $t = 0, 1, \dots$ **do**
- 3: Compute $\nabla f(\mathbf{w}^t) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{w}^t)$
- 4: Distribute $\nabla f(\mathbf{w}^t)$ to all machines
- 5: For each node $k \in \{1, \dots, K\}$, solve

$$\begin{aligned} \mathbf{w}_k^{t+1} = \arg \min_{\mathbf{w} \in \mathbb{R}^D} & \left(F_k(\mathbf{w}) - (\nabla F_k(\mathbf{w}^t) - \eta \nabla f(\mathbf{w}^t))^T \mathbf{w} \right. \\ & \left. + \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}^t\|^2 \right) \end{aligned} \quad (39)$$

- 6: Compute $\mathbf{w}^{t+1} = \frac{1}{K} \sum_{k=1}^K \mathbf{w}_k^{t+1}$
 - 7: **end for**
-

- DANE; Distributed Approximate Newton

- 5 行目では、各ノードが次の部分問題を解いている

$$\mathbf{w}_k^{t+1} = \arg \min_{\mathbf{w} \in \mathbb{R}^D} \left(F_k(\mathbf{w}) - (\nabla F_k(\mathbf{w}^t) - \eta \nabla f(\mathbf{w}^t))^T \mathbf{w} + \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}^t\|^2 \right)$$

- この部分問題の解を得るためのアルゴリズムは、特に指定されていない (何でもよい)
- 問題を解くうえで、他のノードと通信する必要がない (通信コストを十分に小さくできる)
- 各ノードが、摂動を加えた最適化問題を解くアルゴリズムの 1 つ
 $\Leftarrow \mathbf{a}_k^t$ を、 $\mathbf{a}_k^t = \nabla F_k(\mathbf{w}^t) - \eta \nabla f(\mathbf{w}^t)$ と定義している
 $\Leftarrow \mathbf{w}^t \rightarrow \mathbf{w}^*$ ならば $\nabla f(\mathbf{w}^t) \rightarrow \nabla f(\mathbf{w}^*) = 0$ であるので、
 $\mathbf{a}_k^t \rightarrow \nabla F_k(\mathbf{w}^*)$ が成立

- DANE; Distributed Approximate Newton
 - このアルゴリズムは、Federated Learning のために必要な条件 (2) と (3) を満たさない
 - $\mu = 0, \eta = 1$ ならば、条件 (4) を満たす
 - 任意の μ, η について、条件 (1) を満たす
- このアルゴリズムでは、関数が 2 回微分可能であること、各ノードが独立同分布な標本を得られることを仮定
- 正則化パラメータ μ の決め方については、改善の余地がある
 - ⇐ $\mu = 0$ であれば、ノード数 K が小さいときは速やかに収束するが、 K が増えるにつれて、急速に発散しやすくなる
 - ⇐ μ を大きくすれば、アルゴリズムは安定する (発散しづらくなる) が、その分パラメータの収束は遅くなる

- DANE と SVRG を融合したアルゴリズム
 - DANE アルゴリズムは、Federated Learning に適用できない (必要な条件を満たさない)
 - 部分問題の最適解を得る必要がある
 - ⇐ 簡単な問題なら可能だが、複雑な問題であれば計算コストが掛かり過ぎる
 - ⇒ 完全な最適解を得るのではなく、近似解を得るようなアルゴリズムに置き換える
 - ⇒ 部分問題を解くアルゴリズムとして、先程の **SVRG** を使用
 - SVRG アルゴリズムでは、外側のループの最初で、完全な勾配 $\nabla f(\mathbf{w}^t)$ を計算する必要があった (2 行目)
 - 完全な勾配 $\nabla f(\mathbf{w}^t)$ は、各ノードが部分問題を解く前の段階で、既に求まっている (3 行目)
 - ⇒ 各ノードでは、SVRG アルゴリズムの内側のループのみが実行され (後述)、完全な勾配を求める必要はない (既知である)

Algorithm 4: SVRG; Stochastic Variance Reduced Gradient (Recall) [1]

parameters: m = number of stochastic steps per epoch, h = stepsize
(learning rate)

- 1: **for** $s = 0, 1, \dots$ **do**
 - 2: Compute and store full gradient $\nabla f(\mathbf{w}^t) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{w}^t)$
 // Full pass through data
 - 3: Set $\mathbf{w} = \mathbf{w}^t$
 - 4: **for** $t = 1$ **to** m **do**
 - 5: Pick $i \in \{1, \dots, N\}$, uniformly at random
 - 6: Update using $\mathbf{w} = \mathbf{w} - h [\nabla f_i(\mathbf{w}) - \nabla f_i(\mathbf{w}^t) + \nabla f(\mathbf{w}^t)]$
 // Stochastic update
 - 7: **end for**
 - 8: $\mathbf{w}^{t+1} = \mathbf{w}$
 - 9: **end for**
-

Algorithm 5: DANE; Distributed Approximate Newton (Recall) [?]

input : regularizer $\mu \geq 0$, parameter η (default: $\mu = 0, \eta = 1$)

- 1: Initialize \mathbf{w}^0
- 2: **for** $t = 0, 1, \dots$ **do**
- 3: Compute $\nabla f(\mathbf{w}^t) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{w}^t)$
- 4: Distribute $\nabla f(\mathbf{w}^t)$ to all machines
- 5: For each node $k \in \{1, \dots, K\}$, solve

$$\begin{aligned} \mathbf{w}_k^{t+1} = \arg \min_{\mathbf{w} \in \mathbb{R}^D} & \left(F_k(\mathbf{w}) - (\nabla F_k(\mathbf{w}^t) - \eta \nabla f(\mathbf{w}^t))^T \mathbf{w} \right. \\ & \left. + \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}^t\|^2 \right) \end{aligned} \quad (40)$$

- 6: Compute $\mathbf{w}^{t+1} = \frac{1}{K} \sum_{k=1}^K \mathbf{w}_k^{t+1}$
 - 7: **end for**
-

- DANE と SVRG との関係性

- DANE アルゴリズムは、ある特定の条件下では、分散化されたバージョンの SVRG アルゴリズムと等価
 - 以下の 2 つのアルゴリズムは等価である
 - 従って、下記の 2 つのアルゴリズムは、同一のパラメータ列 $\{w^t\}$ を出力する
- 1 $\mu = 0, \eta = 1$ の下で DANE を実行し、その部分問題は SVRG アルゴリズムで解く
 - 2 分散化されたバージョンの SVRG アルゴリズムを解く (後述)
- 分散化された SVRG アルゴリズムを次に示す

Algorithm 6: Naive Federated SVRG (FSVRG) [?]

parameters: m = number of stochastic steps per epoch, h = stepsize,
data partition $\{\mathcal{P}_k\}_{k=1}^K$

- 1: Initialize \mathbf{w}^0
- 2: **for** $t = 0, 1, \dots$, **do**
- 3: Compute $\nabla f(\mathbf{w}^t) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{w}^t)$, distribute to all machines
- 4: **for** $k = 1$ to K **do in parallel** over nodes k **do**
- 5: Initialize $\mathbf{w}_k = \mathbf{w}^t$
- 6: **for** $s = 1$ to m **do**
- 7: Sample $i \in \mathcal{P}_k$ uniformly at random
- 8: Update using $\mathbf{w}_k = \mathbf{w}_k - h [\nabla f_i(\mathbf{w}_k) - \nabla f_i(\mathbf{w}^t) + \nabla f(\mathbf{w}^t)]$
- 9: **end for**
- 10: **end for**
- 11: Update using $\mathbf{w}^{t+1} = \mathbf{w}^t + \frac{1}{K} \sum_{k=1}^K (\mathbf{w}_k - \mathbf{w}^t)$
- 12: **end for**

[1] Pradeep Ravikumar.

Stochastic optimization methods.

http://www.cs.cmu.edu/~pradeepr/convexopt/Lecture_Slides/stochastic_optimization_methods.pdf, 2017.

[2] Suzuki Taiji.

機械学習におけるオンライン確率的最適化の理論.

<https://www.slideshare.net/trinmu/stochasticoptim2013>, 2013.