DEIM2021 A14-3

リッチクライアント-エッジサーバ間での分散機械学習に関する一検討

高野　紗輝† 　中尾　彰宏†† 　山本　周†† 　山口　実靖††† 　小口　正人†

† お茶の水女子大学　〒 112–8610 東京都文京区大塚 2-1-1
†† 東京大学　〒 113–8654 東京都文京区本郷 7-3-1
††† 工学院大学　〒 163-8677 東京都新宿区西新宿 1-24-2
E-mail: †{g1720525,oguchi}@is.ocha.ac.jp, ††nakao@nakao-lab.org, shu@iii.u-tokyo.ac.jp,
†††sane@cc.kogakuin.ac.jp

あらまし　近年 IoT デバイスが普及し, 低遅延やネットワークの負荷分散が可能といった利点を持つエッジコンピューティングに注目が集まっている. 現在行われているこの分野の研究では, デバイス側はセンシングと通信を行うだけの位置付けとなっているものが多い. 一方で, 高性能な CPU や GPU を搭載したデバイスが登場し, これらのリッチクライアントを用いてデータの発生源であるデバイス側で処理を行うと, 通信にかかるコストの削減やプライバシの保護が可能といった点で優れたシステムを構築できると考えられる. そこで, IoT デバイス上で機械学習を行い, その場で結果を利用できるようにすると同時に IoT デバイスで行った学習の続きをエッジサーバで行うモデルの検討を行う. 本稿では, 提案モデルにおいてプライバシや通信コスト, 負荷分散の点で利点があることを確認した.

キーワード　エッジコンピューティング, 分散機械学習, IoT デバイス

1 はじめに

Internet of Things (IoT) の普及によりエッジデバイスから生成されるデータが年々増大しており, 今後 IoT を用いたサービスが普及するにつれ, さらなる増加が予想される. さらに, スマートフォン上に提示されるおすすめ表示や画像認識などと様々な場面で機械学習が活用されるようになった. その結果, エッジデバイスで発生する大量のデータに対して低遅延で複雑な処理を行うことが求められるようになった.

現在主流となっているクラウドコンピューティングではユーザは地理的に遠く離れたクラウドにデータを送信し, クラウド内で処理された結果を応答として受け取る. しかし, エッジデバイス-クラウド間の遅延は数百ミリ秒に及ぶ場合があり, 帯域も多く必要とするため, リアルタイムアプリケーションや大量のデータを送受信するアプリケーションの実装には向いていない.

そこで考案されたのが, エッジデバイスに近いネットワークエッジにエッジサーバを配置して処理を行うことにより, 低遅延でサービスを提供できるエッジコンピューティングである [4]. エッジコンピューティングは, 遠隔にあるクラウドのサーバと比較して物理的に近い位置で処理を行うことにより, 往復の通信遅延時間が低減され, ユーザの体感する応答性や操作性を向上させることが可能となる. また, エッジサーバで分散処理を行うことにより, エッジ-クラウド間におけるネットワークの負荷を削減することも可能である.

一方で, エッジコンピューティングの課題の一つにエッジデバイスが収集した生データの取り扱い方がある. 生データを機械学習処理のために収集元であるエッジデバイスからエッジサーバへと送信すると, データをエッジサーバなどデバイスの外部へと受け渡すことによるプライバシの問題や通信コストが高くなるという問題があり, ユーザ認証プロトコルの導入 [2] やエッジデバイス上でのデータの圧縮・特徴量の抽出 [8] などが考えられている. このように従来のエッジコンピューティングの研究においてエッジデバイス上でのデータの加工は考えられているものの, 性能の低いエッジデバイス側はあくまでデータを収集し, そのデータをエッジサーバに転送するという役割を果たしてきた.

しかし, エッジデバイスの性能向上により, CPU や GPU が搭載され, エッジデバイス内でも機械学習を動かすことのできる程の性能を持つリッチクライアントが登場したことで, より複雑なタスクもエッジデバイス上で実行することが可能となった. 生データではなくエッジデバイス上で行った学習の結果をエッジデバイス-エッジサーバ間で受け渡すことでプライバシや通信コストの問題の改善につながることが期待できる.

そこで, 本研究ではエッジデバイス上で機械学習を行い, その学習結果をエッジサーバと共有することでさらに学習を進め, より良い学習結果を得ることができる分散機械学習モデルを検討する.

本稿の構成は以下の通りである. 第 2 章で関連研究として提案モデルの元のアイディアとなるエッジ/フォグコンピューティングおよびリッチクライアントを用いた分散機械学習の例として Federated learning を紹介する. 第 3 章でエッジコンピューティングの課題について述べ, 第 4 章で解決手法を提案する. 第 5 章で提案モデルの実装と評価を行い, 第 6 章でまとめる.

2 関連研究

2.1 エッジ/フォグコンピューティング

エッジコンピューティングとは, ネットワークエッジにエッジサーバを配置し, データ処理を最大限エッジで行うコンピューティングモデルである. エッジコンピューティングの利点としては, クラウドコンピューティングと比較して低遅延である点やエッジデバイスで処理を行うことでクラウドサーバにかかる負荷を分散できる点, エッジデバイスからクラウドサーバへ送信されるデータ量を削減し, トラフィックの混雑を解消できる点が挙げられる.

論文 [6] ではエッジコンピューティングと似たモデルであるフォグコンピューティングについて一般的なモデルとアーキテクチャについて分析し, クラウドコンピューティングでは数十億のデバイスとクラウド間の長距離通信には通信遅延と帯域幅の圧迫という 2 つの問題が生じるが, クラウドで処理していたタスクをネットワークエッジに設置したフォグサーバにオフロードすることで解決されることが示されている.

このような利点を活かし, エッジ/フォグコンピューティングはスマートシティ [1] や高度道路交通システム [3] などで IoT アプリケーションに応用され, クラウドコンピューティングでは実装することができなかったリアルタイムに応答するシステムが構築されている.

2.2 Federated learning

近年, デバイスの性能向上により, 高性能な CPU や GPU を搭載したリッチクライアントが登場し, IoT デバイス上でサーバが行なっていた機械学習の処理を実行できるようになった. デバイス上で機械学習を行うモデルとして, Federated learning(連合学習) という分散型機械学習が提案された [5]. Federated learning では, まずクラウド上のデータで学習を行って得られた学習モデルを各デバイスに配布し, 各デバイスはそれぞれが収集した固有のデータを利用してさらに学習を進めた上で変更点の情報のみをクラウドに送信する. そして, クラウドは各デバイスから収集した変更点を平均化し, 元の学習モデルを改善してより良いモデルを作成する. このように各デバイスで収集したデータをデバイスの外部に受け渡さないため, プライバシを担保しつつデバイスにあるデータを機械学習に活用することが可能となる. Federated learning はエッジコンピューティングとは異なり, プライバシに配慮しながらエッジデバイスの情報をクラウドに集約し, クラウドが一括管理するコンピューティングモデルとなっている.

論文 [7] では, Federated learning を Google キーボードに応用した例が実装されており, デバイスの持つ固有のデータを受け渡すことなく, デバイス-クラウド間にまたがる分散機械学習が可能であることが示されている.

3 研究課題

エッジコンピューティングにおいて機械学習を行う際に課題となっている問題について以下に示す.

（1） プライバシの保護が必要である.

エッジデバイスで収集したデータはほとんどの場合エッジデバイスのみが所有権を有するものであるため, ネットワークを介した通信やエッジサーバとのデータ共有のためにはプライバシを保護できるセキュリティを確保する必要が出てくる. この問題の解決方法として, エッジデバイスの外部へと送信する情報を学習の重みのみにすることができると, その情報から元のデータを復元することが難しいため, 容易にプライバシの保護が可能となる. しかし, 従来の研究で想定されている性能の低いエッジデバイスでは機械学習を動かすことができないため, このような実装はなされていない.

（2） データをエッジサーバに渡すことができない場合には学習を行うことができない.

従来のエッジコンピューティングモデルでは, 機密性が高くエッジデバイスの外へと持ち出したくないデータを用いたい場合やエッジデバイスが一時的にネットワークに繋がっていない場合, 取得データを即時的に活用したい場合に対応することができない. この問題は, 必要に応じてエッジサーバへ情報を送るか送らないか判断して選別することができる形を作ることで解決することができるが, 従来のエッジサーバやクラウドが学習結果を集約するモデルでは実装が難しい.

（3） 通信コストの削減が必要である.

従来のエッジコンピューティングではエッジデバイス-エッジサーバ間において生データやそれに少し加工を施した容量の大きなデータの受け渡しが行われいる. IoT アプリケーションの普及に対応するためにはネットワークを介して受け渡されるデータに対するさらなる工夫が必要である.

（4） エッジサーバの負荷を分散する必要がある.

1 台のエッジサーバに接続されるエッジデバイスの台数はクラウドコンピューティングの際と比較すると少ないものの, IoT デバイスの普及により場合によっては数千から数万台となり, エッジサーバに大きな負荷がかかることで遅延が発生すると予想される. 従来のエッジコンピューティングにおいてエッジサーバが行っていた学習の一部をエッジデバイスにオフロードすることでエッジサーバにかかる負荷を低減することが望ましいと考えられる.

（5） 各エッジデバイスの持つデータに適した学習結果でない可能性がある.

既存研究では全てのエッジデバイスに対して, クラウドやエッジサーバ上の一般的なデータで学習を行い高精度となった同一のモデルを配布している. しかし, エッジデバイスはエッジサーバと異なる特有なデータを持つ場合があり, 配布された一般的なモデルでは精度が低い可能性がある.

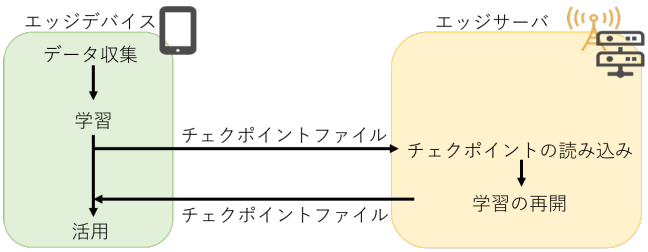

図 1　提案モデル

表 1　エッジサーバの性能

OS	Ubuntu 18.04 LTS
CPU	Intel Core i7-8700
GPU	GeForce RTX 2080Ti
Memory	32Gbyte

表 2　Jetson Nano の性能

OS	Ubuntu 18.04 LTS
CPU	Quad-core ARM A57 @ 1.43 GHz
GPU	128-core Maxwell
Memory	4 GB 64-bit LPDDR4 25.6 GB/s

4　解決手法の提案

リッチクライアントの登場により, 機械学習等の複雑な処理もエッジデバイス上で行うことが可能になったことと合わせ, 上記の課題の解決を目指したリッチクライアントに適した分散機械学習モデルを提案する. 具体的には, エッジコンピューティングモデルにおいて, 従来エッジサーバ上で行っていたタスクの一部をエッジデバイスにオフロードすることでエッジデバイス上でも機械学習処理を行う. さらに, そこで得られた学習結果をエッジサーバと共有することで学習を進め, より良い学習結果をエッジデバイスで利用することのできるモデルを構築する.

提案モデルの概要図を図 1 に示す.

まず, エッジデバイスで収集したデバイス固有のデータを用いて学習を行う. ここで得られた結果は, エッジデバイス上で即時に利用することも可能であるが, より精度の高い結果を得たい場合には学習をエッジサーバへと引き継ぐ. この際, エッジサーバには生データは送信せず, エッジサーバで学習を再開させるために最低限必要な学習の重みを保存したチェックポイントファイルを送信する. エッジサーバは, 受け取ったチェックポイントファイルを読み込み, エッジサーバの持つデータで学習を再開する. 得られたチェックポイントファイルをエッジデバイスへと送信することで, エッジデバイスがより学習の進んだ結果を利用できると考えられる.

5　提案手法の実装と評価

5.1　実 験 環 境

実験で使用したエッジサーバの性能を表 1 に, エッジデバイスとして使用した Jetson Nano の性能を表 2 に示す.

Jetson Nano は GPU を搭載した小型 AI コンピュータボードであり, 近い将来, スマートフォンや様々な IoT デバイスがこのような性能を持つことが期待される.

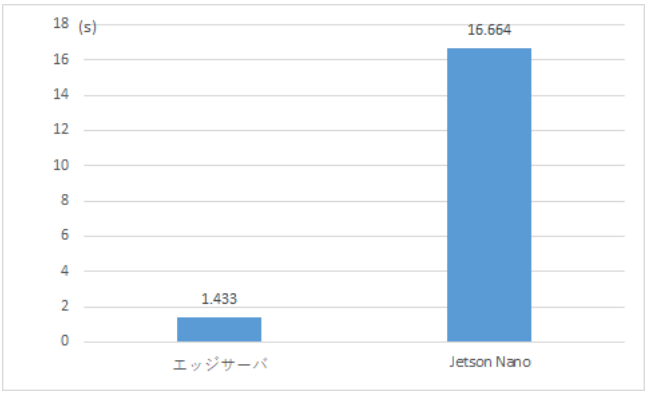

図 2　エッジサーバ, エッジデバイスによる実行時間

また, 本実験では分散処理に適している TensorFlow を機械学習に使用し, Jetson Nano -エッジサーバ間はイーサネットで接続した.

5.2　予 備 実 験

エッジサーバとエッジデバイスにおいて同一の機械学習処理を行った際の実行時間を比較した結果を図 2 に示す.

エッジサーバと比較してエッジデバイスは, およそ 11.6 倍の時間を要する.

一方で, この実験では mnist を epoch 数を 5, 各 step 数を 200 で学習したが, エッジサーバ, エッジデバイス共におよそ 96%と同等精度で学習を行うことが可能である. このことから, エッジデバイスはある程度低速ではあるものの, エッジデバイス内のみでも十分学習できることが分かり, プライバシが非常に重要なデータもそのような形で学習に用いる事が可能になる.

また, エッジサーバでは多数のエッジデバイスと接続することで大きな負荷がかかり, パフォーマンスの低下を引き起こす場合があるが, エッジデバイス内での処理はこの問題が発生しないことも考慮すると, エッジデバイスの性能はかなり高いものだと考えられる.

5.3　実験 1 (均質なデータによる学習)

5.3.1　実 験 概 要

本実験では, 0 から 9 までの手書き数字の画像データセットである MNIST を用いた. 学習を行う前に, エッジデバイスが収集できるデータ量はエッジサーバが保持できるデータ量と比較して少ないと考えられるため, MNIST の 1 から 100 番目のデータをエッジデバイスに与え, それ以降の 101 から 60000 番目のデータをエッジサーバに与えた.

まずはじめにエッジデバイス上で epoch 数を 2, 各 epoch の step 数を 10 で学習を行い, 得られたチェックポイントファイルのみをエッジサーバへと転送する. エッジサーバ側では, 受け取ったチェックポイントファイルを読み込み, epoch 数を 2, 各 epoch の step 数を 1000 で学習を再開させ, 得られたチェックポイントファイルをエッジデバイスへと送信する. ここでは, エッジサーバはエッジデバイスと比べ高性能であり, 短時間で学習

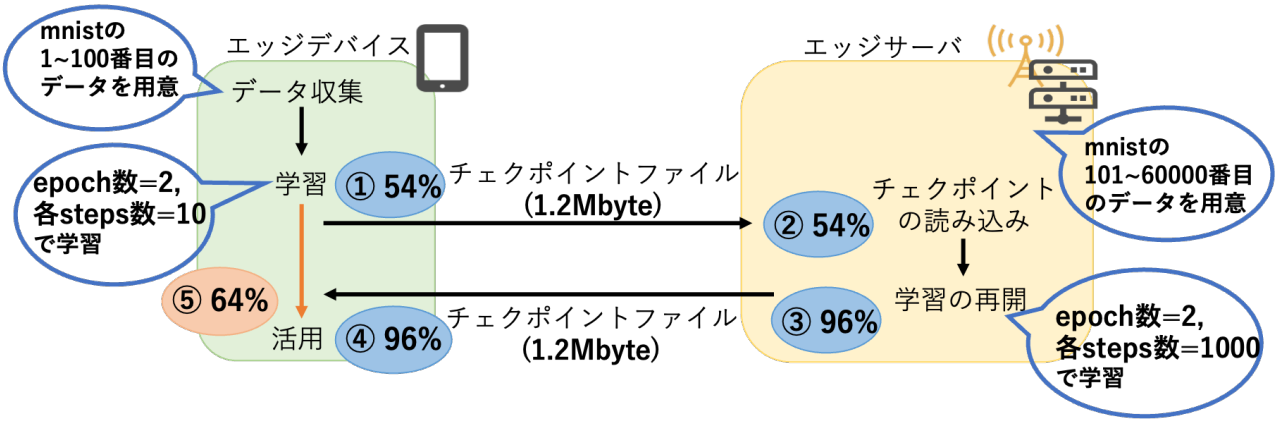

図 3　各ステップでの学習精度

を行うことができるため, より多くの学習を行うようにした.

5.3.2　実験結果

エッジデバイスでの学習後にエッジデバイス上で計測した精度 (①), そこで得られた結果をエッジサーバ上で計測した精度 (②), エッジサーバでさらに学習を行った後にエッジサーバ上で計測した精度 (③), およびその結果をエッジデバイス上で計測した精度 (④) を図 3 に示す. ここでは精度を測るためのテストデータとして, エッジデバイス, エッジサーバ共に MNIST の全てのテストデータ 10000 個を与えた.

エッジデバイス上では①で示すように 54%まで学習できたことが分かり, そこで得られたチェックポイントをエッジサーバへと渡し, エッジサーバ側で計測すると精度は②で示すように同じく 54%となった. よって, エッジデバイスからエッジサーバへの学習処理のオフロードにより, 学習精度は落ちないことが読み取れる. この時点で得られた結果は十分な精度であるとは言えないが, エッジサーバで学習を再開させることで③で示すように 96%の精度を得ることができ, ここで得られたチェックポイントをエッジデバイスに渡し, そこで精度を計測すると④で示すように同じく 96%と高い結果を得ることができた.

また, エッジデバイス上での学習結果をエッジサーバへ引き継いで学習を行い, エッジデバイスにその結果を返す間, エッジデバイス上で引き続き学習を進めると⑤で示すように学習精度は 64%となった. このことから, 提案モデルにおいてエッジデバイスがエッジサーバで学習した結果を得ることで, エッジデバイス上のみで学習した際よりも高い精度の学習結果をその後利用できることが分かる.

さらに, エッジデバイス上での学習およびエッジサーバ上での学習で得られたチェックポイントファイルは共に 1.2Mbyte であった. 一方で, MNIST の画像データは画素数の小さなデータであり, エッジデバイスで用いた 100 個の画像ファイルは 76Kbyte であるが, 画像データ数が増えるとファイルの容量も増加し, MNIST 全体の 60000 個の画像ファイルでは 45Mbyte となる. チェックポイントファイルの容量は学習に使用したデータ数に依らないため, 多くの学習データを使用する場合や容量の大きなデータを学習に使用する場合にはチェックポイントファイルを利用することで通信データ量が削減できるという結果が得られた.

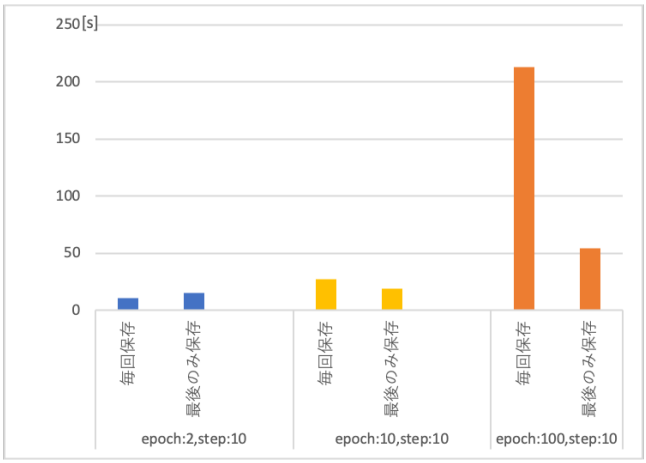

図 4　チェックポイントファイルに 1epoch 実行毎に保存する場合と学習の最後のみ保存する場合の実行時間

5.4　高速化

5.4.1　実験概要

上記の実験では Keras を用いたモデルの保存と復元を実装している. この実装では, 1 epoch 毎に学習の重みをチェックポイントに保存しているが, 提案モデルで学習結果の受け渡し後に使用しているチェックポイントファイルは最後の epoch で保存したファイルのみである. そこで, 最後のみ学習結果をチェックポイントに保存するよう実装を変更する. 具体的には, まずはチェックポイントに学習結果を保存せずに学習を進める. そして, 最後に学習結果をチェックポイントに保存する形で epoch 数 1, step 数 1 で学習を行い, 学習を引き継ぐ際に必要なチェックポイントファイルを作成する.

5.4.2　実験結果

エッジデバイス上においてチェックポイントファイルに各 epoch を実行する毎に保存する場合と, 最後のみ保存する場合の実行時間を epoch 数を変化させて比較し, 図 4 に示す.

実験 1 でエッジデバイス上で学習を行った epoch 数 2, 各 epoch の step 数 10 の学習では, 各 epoch 毎にチェックポイントに保存する場合の方が最後のみチェックポイントに保存する

場合と比較して速いという結果となった．これは，最後に行うチェックポイントを保存する学習のはじめにチェックポイントファイルの初期化等の処理を行っており，そこに時間がかかっているためだと考えられる．一方で，epoch 数を増加させていくと各 epoch 毎に保存する場合よりも最後のみ保存する場合の方が実行時間が短くなり，その差が大きくなっていくという結果が得られた．

本実験では MNIST で学習を行っており，少ない学習で高い精度の結果を得ることができているが，実際にスマートフォン上にあるようなデータで学習を行う際にはより多くの学習が必要であると考えられ，提案する高速化は有効だと言える．

5.5 実験 2 (偏りのあるデータによる学習)

実験 1 においてはエッジデバイス，エッジサーバ共に 1 から 9 の数字をほぼ均等に与えており，エッジデバイスが特徴的なデータを持たず，エッジデバイスとエッジサーバが類似したデータを保持していることを想定している．しかし，ユーザがエッジデバイスで収集したデータの中には，ユーザ特有のデータが含まれている可能性や，データに偏りが生じている可能性がある．すると，学習をエッジサーバへと引き継ぐことで，逆に学習精度が下がってしまうという懸念が生じる．

本実験では，エッジデバイスおよびエッジサーバが保持するデータを変化させ，それが学習結果に及ぼす影響について考察する．

5.5.1 実 験 概 要

実験 1 の実験環境において，エッジデバイスとエッジサーバに与えるデータに以下のように偏りを持たせる．

（1） MNIST の”1”のデータのみをエッジデバイスに与え，エッジサーバには全ての数字のデータを与える．

これはエッジデバイスが特徴的なデータを持ち，エッジサーバはエッジデバイスの持つ特徴的なデータを含め，より多くの種類のデータを保持しているケースである．エッジデバイスの保持できるデータ量や種類はエッジサーバと比較して少ないと考えられるため，このようなデータの偏りが起きることは容易に想定できる．

（2） MNIST の全ての数字のデータをエッジデバイスに与え，エッジサーバには”1”以外のデータを与える．

この場合，エッジデバイスにはエッジサーバの保持しない特有なデータも含まれている．プライバシの観点からエッジデバイスの外へ持ち出せないデータがある場合には，このケースのようにエッジサーバの方がエッジデバイスと比較して学習の際に使用できるデータの種類が限定されてしまうことが考えられる．

（3） MNIST の”1”のデータのみをエッジデバイスに与え，エッジサーバには”1”以外のデータを与える．

これは (2) のさらに極端な場合であり，エッジデバイスとエッジサーバが全く異なる学習データを保持するケースである．

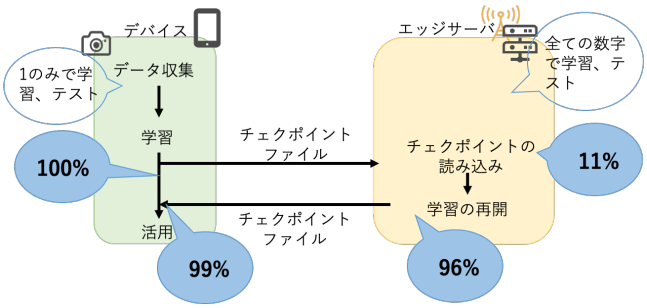

図 5 エッジデバイスに”1”のみ，エッジサーバに全ての数字を与えた際の学習精度

そして，実験 1 と同様にエッジデバイスでは epoch 数を 2，各 epoch の step 数を 10 で学習を行い，エッジサーバでは学習を引き継ぎ，epoch 数を 2，各 epoch の step 数を 1000 で学習を行う．

5.5.2 実 験 結 果

上記で示したそれぞれについて以下で結果をまとめる．

（1） エッジデバイス：”1”のみ，エッジサーバ：全ての数字

各状態での学習精度を図 5 に示す．

エッジデバイス上において”1”のみのデータで学習を行い，”1”のみのテストデータで精度を計測すると，すぐに 100%と良い結果を得ることができた．しかし，全ての数字のデータを持つエッジサーバ上で精度の計測を行うと，”1”以外の学習を行っていないため 11%と低い結果となった．エッジサーバ上で全ての数字のデータで 96%まで学習を行う．その結果をエッジデバイスに渡し，”1”のみのテストデータで計測を行うと 99%とエッジサーバに学習をオフロードしてから戻すことで多少精度が落ちるものの，依然として高い精度を得ることができた．

また，エッジサーバ上での学習を増やしても最後にエッジデバイス上で得られる学習精度は 97%〜99%と高い精度の結果を得ることができた．

したがって，エッジデバイスが偏ったデータを持つ場合であっても，エッジデバイスの持つデータと近いデータをエッジサーバが保持している場合にはエッジサーバに学習を引き継ぐことによる学習精度の低下は少なく，提案モデルに問題はない．

（2） エッジデバイス：全ての数字，エッジサーバ：”1”以外

各状態での学習精度を図 6 に示す．

エッジデバイス上において全ての数字のデータで学習することで 54%の精度まで学習が進み，その学習結果の精度をエッジサーバ上の”1”以外の数字のデータで計測すると 44%と下がる結果となった．エッジサーバ上でさらに 96%まで学習を進め，その学習結果をエッジデバイスに渡す．これをエッジデバイス上の全ての数字のデータで精度を計測すると 76%となった．エッジデバイスのみで学習する場合と比較すると精度の向上が見られるものの，高い精度の結果を得ることはできなかった．

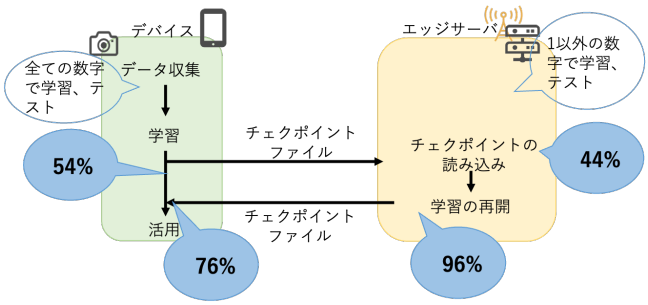

図 6　エッジデバイスに全ての数字, エッジサーバに”1”以外の数字を与えた際の学習精度

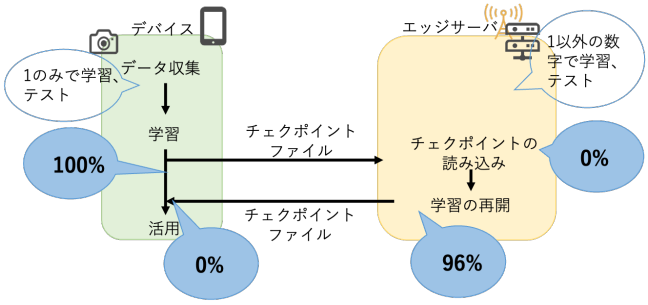

図 7　エッジデバイスに 1 のみ, エッジサーバに”1”以外の数字を与えた際の学習精度

（3）　エッジデバイス：”1”のみ, エッジサーバ：”1”以外

各状態での学習精度を図 7 に示す.

エッジデバイス上の”1”のみで 100%まで学習した結果の精度を”1”のデータを持たないエッジサーバ上で計測すると, 0%と”1”以外の数字のデータには対応できていないことが確認できた. さらに, ”1”以外の数字のデータを用いて 96%の精度まで学習を進めた後, エッジデバイス上の”1”のみのデータで精度を計測すると 0%となった.

したがって, エッジデバイスの持つデータに近いデータがエッジサーバ上に存在しない場合, エッジサーバに学習を引き継ぐことによりはじめにエッジデバイスで学習した結果が失われてしまい, エッジデバイスで使用できる学習結果の精度が下がると考えられる.

5.6　提案モデルの改善

実験 2 の結果より, 図 1 に示した提案モデルにおいて, エッジデバイスがエッジサーバの保持しない特有なデータで学習を行っている際には, エッジサーバへと学習を引き継ぐことにより学習精度が下がってしまう可能性がある. そこで, エッジデバイスとエッジサーバで学習するデータに偏りがある場合にも対応できるモデルとして 図 8 を提案する. エッジデバイス上において, エッジサーバでの学習後に受け取ったチェックポイントファイルとその時点でのエッジデバイスでの学習結果を保存したチェックポイントファイルをそれぞれ読み込む. 精度を比較し, より精度の高い学習結果を利用する.

例えば図 3, 図 5 のようにエッジサーバへ学習をオフロードして精度が上がった場合はその学習結果を用い, 図 7 のようにオフロードして精度が下がった場合は, その結果ではなくエッジデバイスで学習し続けた結果を用いるようにする.

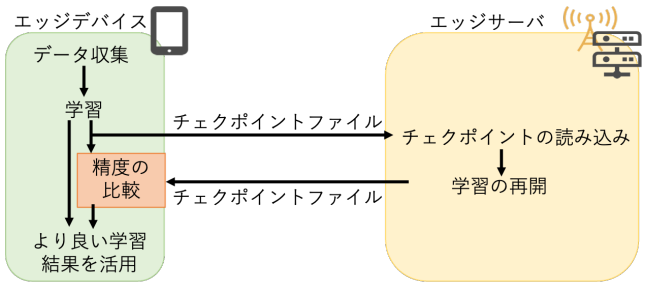

図 8　学習データの偏りに対応した提案モデル

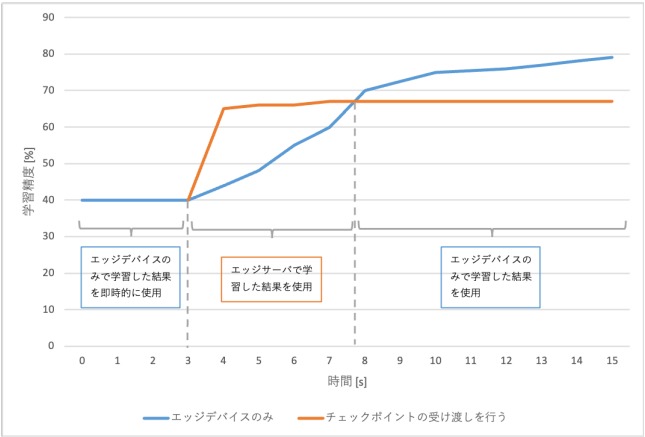

図 9　学習データの偏りに対応したモデルの動作例

このモデルの動作例を図 9 に示す. ここではエッジデバイスに MNIST の全ての数字をほぼ均等に含む 1000 個のデータを与え, エッジサーバに MNIST 全体の”1”および”2”以外の数字のデータを与えた. エッジデバイス, エッジサーバ間におけるチェックポイントファイルの受け渡しにおよそ 3 秒かかるため, その間はエッジデバイス上のみで学習した結果を即時的に使用する. その後エッジサーバ上で学習したより高い精度を持つ学習結果を受け取り, その結果を選択して活用する. しかし, エッジサーバには”1”および”2”のデータが存在しないため, 68%以上の精度を得ることができない. 一方で, エッジデバイスは”1”および”2”も含むデータで学習しているため, 時間はかかるものの 80%以上の精度を得ることが可能である. この例ではおよそ 8 秒の時点でエッジデバイス上のみでの学習結果がエッジサーバで学習した結果よりも精度が高くなるため, それ以降はエッジデバイスのみで学習した結果を選択して利用する.

このモデルでは, サーバに頼ることでより精度の上がった結果を使用することができる. 一方で, エッジサーバに学習を引き継いで精度が落ちてしまった場合には, エッジデバイスのみで学習したより良い精度の結果を利用することができるという利点がある.

6　まとめと今後の課題

従来のエッジコンピューティングで課題となっているプライバシの保護や通信コストの削減を目的として, リッチクライアントに適した分散機械学習モデルの検討を行った.

従来のエッジコンピューティングモデルにおいて, エッジデバイスでも機械学習処理を行い, 学習結果をエッジサーバと共有することでさらに学習を進めるモデルを提案し, エッジデバイス側に Jetson Nano を用いて実験を行なった. その結果, エッジサーバに学習を引き継いで高い精度の結果を得ることができ, 生データをエッジデバイス上に留めておくことでプライバシの保護および通信コストの削減が可能であることが示された. また, エッジデバイスがエッジサーバと異なるデータを持つ際にエッジサーバ上での学習結果を用いることで精度が下がるという問題が生じることを示し, この問題に対応できるよう提案モデルを改善した.

今回は MNIST を用いて実験を行なったが, 今後はスマートフォン上にあるような画像や SNS のテキスト情報を用いて実験を行い, それらを利用するアプリケーションの構築および検討を行いたいと考えている.

文　献

[1] N. Chen, Y. Chen, S. Song, C. Huang, and X. Ye. Poster abstract: Smart urban surveillance using fog computing. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 95–96, 2016.

[2] Junho Lee, Dongwook Kim, Jinhyun Park, and Hyungweon Park. A multi-server authentication protocol achieving privacy protection and traceability for 5g mobile edge computing. *Proc. of the 39th IEEE International Conference on Consumer Electronics (ICCE 2021)*, January 2021.

[3] J. Lin, W. Yu, X. Yang, Q. Yang, X. Fu, and W. Zhao. A real-time en-route route guidance decision scheme for transportation-based cyberphysical systems. *IEEE Transactions on Vehicular Technology*, Vol. 66, No. 3, pp. 2551–2566, 2017.

[4] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys Tutorials*, Vol. 19, No. 3, pp. 1657–1681, 2017.

[5] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications, 2019.

[6] S. Yang. Iot stream processing and analytics in the fog. *IEEE Communications Magazine*, Vol. 55, No. 8, pp. 21–27, 2017.

[7] T. Yang, G. Andrew, Hubert Eichner, Haicheng Sun, W. Li, Nicholas Kong, D. Ramage, and F. Beaufays. Applied federated learning: Improving google keyboard query suggestions. *ArXiv*, Vol. abs/1812.02903, , 2018.

[8] 純塩田, 允滝澤, 裕之田中, 紀之高橋, 英嗣小林. 画像処理におけるエッジコンピューティングを用いた垂直分散処理方式の検討. Technical Report 2, 日本電信電話株式会社, 日本電信電話株式会社, 日本電信電話株式会社, 日本電信電話株式会社, 日本電信電話株式会社, nov 2016.