CF-008

平文上の処理ロジックを再現した秘密計算ディープラーニング
A Secure Deep-Learning Realizing the Processing on Plaintext

三品 気吹 * 濱田 浩気 * 五十嵐 大 *

あらまし 本論文では，データを暗号化したまま学習や予測を行う，秘密計算ディープラーニングの実現を目指す．秘密計算は，結果以外誰もデータを見ないという性質から，複数者のデータを統合してデータ分析を行う新しい分析である，統合データ分析を実現すると見込まれる暗号技術である．現在最もホットであるディープラーニングを秘密計算で再現することができれば，この新しいパラダイムの促進に貢献できるものと考えられる．しかし秘密計算は処理性能に課題があるため，既存の秘密計算上のディープラーニングにおいては (1) 出力層の活性化関数であるソフトマックス関数に必要な指数関数を，近似としては精度が高いとはいえない，ReLU 関数で近似している (2) 収束性を改善する最適化手法を再現できていない，等の課題がある．本論文では (1) に関してソフトマックス関数を，(2) に関して代表的な最適化手法である Adam を，それぞれ秘密計算上で実現し，平文における処理を再現する．さらに，それでも性能は既存実装と比べ約 9 倍であったことを報告する．

1 はじめに

秘密計算とは，データを暗号化したまま計算する技術である．秘密計算を用いることで，複数のデータホルダから，お互いのデータを秘密にしたまま集めることができる．それによって，これまでは実現しなかった自由な統合分析を，情報を安全に守りながら実現できるようになる．その分析手法の一群として，特に注目されているのが機械学習 (AI) であり，秘密計算分野でも活発に研究が行われている．筆者らは，scikit-learn[8] などの既存の AI ライブラリで提供されている分析手法を，秘密計算上で実現していくことを目指している．

それに向けて，筆者らはこれまでに，秘密計算上でのロジスティック回帰分析に取り組んでおり，平文の分析ツールである R と比較しても遜色無い性能を達成した [30, 31]. そして本論文では，AI 分野でも多方面での利活用が特に期待されている，ニューラルネットワークに着目した．ニューラルネットワークには様々な形があるが，本論文では最も基本形の全結合・順伝播型で，隠れ層が 2 層以上あるディープ・ニューラルネットワーク (ディープラーニング) の実現に取り組む．

既存の秘密計算上ディープラーニングにおいては，出力層の活性化関数であるソフトマックスに必要な指数関数を，単純な関数である ReLU 関数で代替としている．論文上の実験ではかなりの精度ではあるがデータセットにより平文上の実装と比べ予測精度に課題が現れることが予想される．

またディープラーニングにおいて，良い学習結果を速く得るための最適化手法 [25] は複数存在するが，それらの殆どは秘密計算が苦手とする計算を含むため，再現がなされていない．最適化手法には，(1) 予測精度の高い学習結果が得られる場合がある，(2) 学習処理の収束が速くなり処理が高速化される，等のメリットがあり，平文上のディープラーニングでは広く利用されている．(1) の性質はもちろんのこと，秘密計算においては (2) の収束高速化効果と最適化手法における処理の複雑さと，どちらが優位になるか，興味深い課題である．

1.1 関連研究

秘密計算ニューラルネットワークの関連研究としては，Barak らが複製秘密分散 [12, 16] を用いて量子化ニューラルネットワークを実装したもの [11] や，Liu らが ABY ライブラリ [13] を用いて実装した MiniONN[20] や，Juvekar らが独自の暗号と garbled circuit[29] を組み合わせて実装した GAZELLE[17] などがある．これらは全て「予測」の部分のみを実装しており，学習の部分は実装されていない．先行研究でニューラルネットワークの学習部分を実装しているものは，以下の 2 つである．

1.1.1 SecureML

SecureML[21] では秘密分散に garbled circuit [29] や紛失通信 [23] といった暗号を組み合わせて，隠れ層が 2 層の全結合・順伝播型ディープニューラルネットワークの学習と予測を実装し，MINIST データ [7] を用いて実験している．2 種類のプロトコルを提案しており，全体の処理時間が短くなるプロトコルでは，隠れ層の活性化関数として ReLU 関数を用いた場合の結果が，オフライン処理時間が 14951 秒，オンライン処理時間が 10332 秒で合計 7 時間程度，予測精度は 93.4%となっている．隠

* NTT セキュアプラットフォーム研究所, 東京都武蔵野市緑町 3-9-11

れ層の活性化関数として $f(x) = x^2$ を用いた場合，合計処理時間が約 6 時間，予測精度は 93.1%となっている．もう 1 つのプロトコルでは，ReLU と二乗関数を用いた場合，それぞれのオンライン処理時間は 4240 秒，653 秒だが，オフライン処理時間が約 80 時間，約 89 時間となっている．

1.1.2 SecureNN

SecureNN[28] では，3 パーティの加法秘密分散を用いてディープニューラルネットの学習と予測を実装し，SecureML と同様に MNIST データで実験を行っている．Eigen[3] という高速な行列演算ライブラリを用いることで高速化し，SecureML と同じ設定の実験では約 1 時間，予測精度は 93.4%という結果になっている．

1.2 本論文の貢献

本論文では，秘密計算ディープラーニングにおいて，2 つの課題を解決する．

(1) ソフトマックス関数の課題に対して，秘密一括写像 [32] を用いて任意の精度でソフトマックス関数自体を近似可能なアプローチをとる．これは，理論的にソフトマックス関数の近似を保証するものであり，理論的にソフトマックス関数の近似とはいえない ReLU 関数を用いた処理とは本質的に異なる改善である．

(2) 最適化手法の課題に対して，最適化手法である Adam を実現し，平文上のディープラーニングを十分な精度で再現し，性能も最適化手法なしの場合よりも向上したことを報告する．

1.3 AI とディープラーニング

AI の利活用が期待されている領域は多岐にわたり，マーケティング，医療，運輸，製造業など様々な領域で，AI を用いた予測や業務効率化などが検討されている [22]．その中でも，ディープラーニングは他の AI 手法では難しかった分析も実現できるとして，医療 [2, 4] や金融 [9] などの領域で実用化されている．例えば，レントゲン写真や MRI などの画像から病気の可能性を判定したり [4]，病気を引き起こす遺伝子変異を発見したり [2]，ソーシャルデータや電話代の支払い履歴などから「信用」の判定を行う [9] といったことにディープラーニングが利用されている．

1.4 秘密計算 AI

AI を利活用するうえで，より良い結果を得るために多くの学習データが必要だったり，企業によっては自社データのみでは偏りがあるために，他社データも利用したいといった声がある [36, 37]．しかし，これまでは共同実験契約を結ぶ等の限られた条件下でしか，企業を跨いでデータを利用する統合分析は行われてこなかった．

また，AI 利活用が活発になった背景の 1 つとして，個人の IoT デバイスや携帯端末等から，移動情報・購買履歴といった多くのパーソナルデータが収集可能になったことがあるが，そのような情報を利用するならば，プライバシーの問題も発生する．特に日本では個人データを扱うことに関して，リスクや社会的責任の大きさを強く感じており，データ利活用の障壁となっている [33]．

そこで，AI 利活用促進に対する阻害要因の 1 つとなっている，個人や企業のプライバシーに関するリスクを低減しつつ，これまでより自由な AI 利活用を可能にするための技術として，秘密計算の活用が活用できる．秘密計算の「データを暗号化したたまま集めて計算できる」という性質を，AI と組み合わせることによって得られるメリットを 2 つ示す．この 2 つのメリットは，機械学習の「学習」と「予測」の各フェーズにそれぞれ対応し，単体でも組み合わせても利用できる．

1.4.1 学習データを秘匿したモデル構築

学習データを秘匿したまま複数の企業・個人からデータを集め，新たな AI モデルを作ることができる．

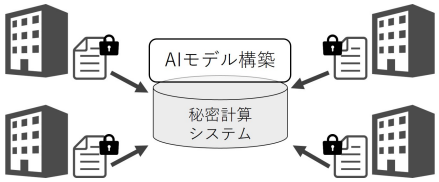

図 1.1: 学習データを秘匿したモデル構築

学習が終わった時点のパラメータも暗号化されている．学習済みのパラメータは復号して利用することも，そのまま暗号化した状態で利用することも可能である．

1.4.2 モデルと入力を秘匿した予測

学習済みのモデルと，そこへ入力する情報及び予測結果を秘匿することができる．

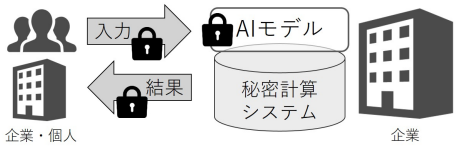

図 1.2: モデルと入力を秘匿した予測

2 準備

2.1 記法

ベクトルを $\vec{a} := (a_0, \ldots, a_{n-1})$ と書き．a を b で定義することを $a := b$ と書き，同じ要素数の 2 つのベクトル $\vec{a}, \vec{b}$ の内積を $\vec{a} \cdot \vec{b}$ と書く．また，$i \times j$ の行列を $a_{i,j}$ と書

き，2 つの行列の積を $(\cdot)$ と書き，2 つの行列の要素ごとの積を $(\circ)$ と書く．$(a_{i,j})^T$ は $a_{i,j}$ の転置行列を表す．演算子が書かれていないものはスカラー倍である．

2.2 ニューラルネットワーク

ニューラルネットワークには様々な構造が存在するが，本論文では図 2.1 に示すような全結合・順伝播型のニューラルネットワークを用いる．

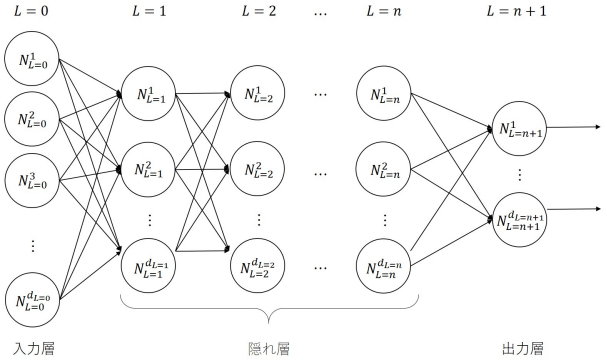

図 2.1: ディープニューラルネットワーク

L は層の番号を示し，隠れ層の数が n 層の時，入力層は $L=0$，出力層は $L=n+1$ となる．N_j^i は $L=j$ 層目の i 番目のニューロンを示す．$d_{L=j}$ は j 層目のニューロンの数を示す．また，各ニューロンの繋がりの強さをパラメータ (重み) と呼び，適切な出力を与えるパラメータの値を推定することを「学習」と呼ぶ．

2.2.1 活性化関数

ニューラルネットワークでは，隠れ層と出力層の活性化関数をそれぞれ目的に応じて選択する．本論文では下記の 2 つの活性化関数を用いる．

ReLU 関数　ディープニューラルネットワークでは，中間層の活性化関数として非線形な関数を用いる．その中でも ReLU 関数は，深いネットワークでも良い学習結果を得られることが知られており，ディープラーニングの分野で頻繁に使われている [14][24]．ReLU 関数は式 (2.1) に示すような，入力が正であればそのままの値，入力が負であれば 0 を返す関数である．このように条件分岐のみで実現できるという，秘密計算との相性の良さからも，本論文では ReLU 関数を隠れ層の活性化関数として用いる．

$$\mathsf{ReLU}(u) = \mathsf{max}(0, u) \tag{2.1}$$

ReLU は微分不可能な関数だが，計算機的には下記のような関数で微分 ReLU′ を計算する．

$$\mathsf{ReLU}'(u) = \begin{cases} 0(u \leqq 0) \\ 1(u > 0) \end{cases} \tag{2.2}$$

ソフトマックス関数　ニューラルネットワークでは，解きたい問題に合わせて出力層の活性化関数を選択する．多クラス分類の問題を解く場合は，出力層にソフトマックス関数 $\mathsf{softmax}(u_i)$ を用いるのが一般的である．k 個のクラスに分類する場合のソフトマックス関数は以下のようになる．

$$\mathsf{softmax}(u_i) = \frac{\exp(u_i)}{\sum_{j=1}^{k} \exp(u_j)} \tag{2.3}$$

また，出力層にソフトマックス関数を用いる場合，誤差関数にはクロスエントロピー誤差を利用する．学習データの正解ラベルを $t_{m,k}$，現在のモデルからの出力をを $y_{m,k}$ とすると，クロスエントロピー誤差 $E(w)$ は下記のようになる．

$$E(w) = -\sum_m \sum_k t_{m,k} \circ \log y_{m,k} \tag{2.4}$$

誤差関数の u_k での偏微分は次のようになる．

$$\frac{\partial E_m(w)}{\partial u_k} = y_{m,k} - t_{m,k} \tag{2.5}$$

2.2.2 バックプロパゲーション

隠れ層のあるニューラルネットワークの学習にはバックプロパゲーションという手法が用いられる [26]．また本論文では，一般的なバックプロパゲーションにおけるテクニックに従い，ミニバッチ確率的勾配降下法を採用する [19]．以下に隠れ層の活性化関数に ReLU 関数，出力層の活性化関数にソフトマックス関数を用いたバックプロパゲーションで，パラメータ (重み) を学習する処理の流れを示す．w^{l-1} は $l-1$ 層目と l 層目の間のパラメータを表し，y_i^l は l 層目のニューロンのうち i 番目からの出力を表す．

m は 1 度の学習に用いるデータの数，すなわちバッチサイズである．全データを 1 回使いきることを 1epoch と表現し，全データ数 M，バッチサイズ m の場合，1epoch 中で $\frac{M}{m}$ 回のミニバッチによる学習が行われる．

- 入力層から出力層に向かって順伝播を計算
  $i = d_{l-1}, j = d_l$

$$u_{m,j}^l = y_{m,i}^{l-1} \cdot w_{i,j}^{l-1} \tag{2.6}$$

$$y_{m,j}^l = \mathsf{ReLU}(u_{m,j}^l) \tag{2.7}$$

- 出力層の誤差 $z_{m,k}^{n+1}$ を計算

$$z_{m,k}^{n+1} = y_{m,k}^{n+1} - t_{m,k} \tag{2.8}$$

- z^l を逆伝播させ，各層の z^{l-1} を計算

$$z_{m,i}^{l-1} = \mathsf{ReLU}'(u_{m,i}^{l-1}) \circ (z_{m,j}^{l} \cdot (w_{i,j}^{l-1})^T) \tag{2.9}$$

- 各層の勾配 g^l を計算し，パラメータを更新する

$$g_{i,j}^{l-1} = (y_{m,i}^{l-1})^T \cdot z_{m,j}^{l} \tag{2.10}$$

$$w_{i,j}^{l-1} = w_{i,j}^{l-1} - \frac{\eta}{m} g_{i,j}^{l-1} \tag{2.11}$$

2.2.3 学習アルゴリズムの最適化：Adam

単純な勾配降下法は式 (2.11) でパラメータを学習する方法である．この方法は比較的実装が容易だが，局所解に捉まりやすい・収束が遅いなどの問題が知られている．それらを解決するために様々な最適化手法が提案されており，代表的な 8 手法が [25] で紹介されている．

最適化を施すことで，収束のスピードを速めたり，局所解に捉まってしまうのを防ぐことができるため，実際の深層学習フレームワークでも利用されている [1, 5, 6]．本論文では，[25] で特に良い学習結果を得やすいと紹介されていた Adam を用いる．単純な勾配降下法と Adam で，式 (2.10) までの処理は同じである．

以下に Adam の処理を示す．なお，各層での処理は同じため層番号 l は省略し，何回目の学習かを表す変数 t を加え，g_t は t 回目の勾配を表す．また，$m, v, \hat{m}, \hat{v}$ は g と同じ大きさの行列であるため，行列の大きさを表す添え字は省略し，全て 0 で初期化しているものとする．ここでの上付き数字 t は t 乗を表す．

$$m_{t+1} = \beta_1 m_t + (1-\beta_1) g_t \tag{2.12}$$

$$v_{t+1} = \beta_2 v_t + (1-\beta_2) g_t \circ g_t \tag{2.13}$$

$$\hat{m}_{t+1} = \frac{1}{1-\beta_1^t} m_{t+1} \tag{2.14}$$

$$\hat{v}_{t+1} = \frac{1}{1-\beta_2^t} v_{t+1} \tag{2.15}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_{t+1}} + \epsilon} \circ \hat{m}_{t+1} \tag{2.16}$$

β_1, β_2 は 1 に近い定数で，Adam の提案論文では $\beta_1 = 0.9, \beta_2 = 0.999$ となっている．また η は学習率で，提案論文では $\eta = 0.001$，ϵ は $\sqrt{\hat{v}_{t+1}} = 0$ の場合に式 (2.16) が計算できないのを防ぐための値で，Adam の提案論文では $\epsilon = 10^{-8}$ となっている [18]．

2.2.4 He の初期値

パラメータの初期化方法は活性化関数などに合わせて選択するのが良く，中間層の活性化関数に ReLU 関数を用いる場合は，He らが提案した初期化方法だと良い学習結果を得やすいことが知られている [15]．本論文では He の初期値を採用し，具体的には L_{j-1} 層目と L_j 層目の間のパラメータを，範囲 $[-\sqrt{\frac{6}{d_{j-1}}}, \sqrt{\frac{6}{d_{j-1}}}]$ の一様分布を用いて初期化する．

2.3 秘密計算

2.3.1 秘密分散を用いた秘密計算

秘密計算にはいくつかの方式があり，その中でも秘密情報を「シェア」と呼ばれる幾つかの断片に変換する秘密分散方式を用いたものは，データの処理単位が小さく高速に処理できることが知られている [10, 34]．秘密分散の具体的な構成方法は Shamir の秘密分散 [27] や複製秘密分散 [12, 16] などが知られており，本論文では n 個のシェアを生成し，k 個以上のシェアからは秘密が復元できるが，k 個未満のシェアからは秘密の情報が全く漏れない (k, n) 閾値法という秘密分散を用いる．

2.3.2 プログラマブルな秘密計算ライブラリ MEVAL

MEVAL 筆者らが開発する秘密分散ベースの秘密計算ライブラリで，加算・乗算・ソートなど 100 以上の演算を組み合わせて自由にプログラムできる [35]．MEVAL は演算に応じて複数の秘密分散を用いているが，本論文では主に $\mathrm{GF}(2^{61}-1)$ 上の Shamir 秘密分散を用い，そのシェアを $[\![\vec{a}]\!]$ と書く．

シェアの定数倍や加算は通信無しで行うことができ，これを $a[\![\vec{b}]\!]+c = [\![a\vec{b}+c]\!] = (ab_0+c, ab_1+c, \ldots, ab_{m-1}+c)$ などと書く．このようにベクトルの要素ごとに同じ演算を行う場合，ベクトルの単一要素への演算と同様の記法をとる．MEVAL で用意されている演算のうち，バックプロパゲーションを実装するうえで特に重要な演算を以下に示す．処理単位はベクトルである．

- map: 秘密一括写像. シェアの列 $[\![\vec{a}]\!] := ([\![a_0]\!], \ldots, [\![a_{m-1}]\!])$ と定義域および値域 $(x_0, \ldots, x_{\ell-1})$, $(y_0, \ldots, y_{\ell-1})$ を入力とし，各入力値を写像させたシェア，すなわち $0 \le i < m$ について $x_j \le a_i < x_{j+1}$ 且つ $b_i = y_j$ であるような $([\![b_0]\!], \ldots, [\![b_{m-1}]\!])$ を出力する．

- rshift: シェアの列 $[\![\vec{a}]\!] := ([\![a_0]\!], \ldots, [\![a_{m-1}]\!])$ と公開値 t を入力とし，$[\![\vec{a}]\!]$ の各要素を t[bit] 算術右シフトした $[\![\vec{b}]\!] := ([\![b_0]\!], \ldots, [\![b_{m-1}]\!])$ を出力する．算術右シフトは左側を 0 ではなく符号ビットでパディングするシフトであり，論理右シフト rlshift [31] を用いて以下のように $\mathsf{rshift}([\![A \times 2^n]\!], n-m) = [\![A \times 2^m]\!]$ を構成する．

$$[\![A' \times 2^n]\!] = [\![A \times 2^n]\!] + a \times 2^n (a \geq |A|) \quad (2.17)$$
$$[\![A' \times 2^m]\!] = \mathsf{rlshift}([\![A' \times 2^n]\!], n-m) \quad (2.18)$$
$$[\![A \times 2^m]\!] = [\![A' \times 2^m]\!] - a \times 2^m \quad (2.19)$$

3 提案手法

秘密計算でバックプロパゲーションを実現するにあたり，以下に示す秘密計算の苦手な処理を，如何に効率良く行うかが非常に重要である．

- 浮動小数点数
- 除算や指数関数の計算

この 2 点に対する対処法と，バックプロパゲーションを秘密計算で行う秘密計算 back prop アルゴリズムを示す．

3.1 固定小数点数によるアルゴリズム設計

秘密計算では浮動小数点数の処理コストが大きいため，本論文では固定小数点数を用いてアルゴリズム設計を行う．変数や定数ごとに精度を設定しており，精度が α[bit] の場合，秘密情報 $[\![x]\!]$ は基数 2 の固定小数点数 $[\![x \times 2^\alpha]\!]$ として秘密分散されている．

全ての変数・定数の精度を同じにしない理由は，それぞれの変数や定数ごとに値域がことなり，必要な精度が異なるためである．例えばパラメータは小さくなりやすいため，精度を他の値よりも精度を高めに設定しておくといった設計をしている．

また固定小数点数で処理を行う場合，乗算を行った際に精度が変わることが問題となる．たとえば $[\![x_1 \times 2^\alpha]\!] \times [\![x_2 \times 2^\alpha]\!]$ という乗算を行うと，結果は $[\![x_1 x_2 \times 2^{2\alpha}]\!]$ となり，元の値よりも α[bit] 大きくなってしまう．バックプロパゲーションでは層ごとに乗算を行うため，普通に処理すると途中でオーバーフローしてしまう．そこで本論文では，右シフト (rshift)[31] を用いることで，意図的に桁を落としてオーバーフローを防ぎつつ，固定小数点数で効率良くバックプロパゲーションを実現する．本論文では表 1 のように精度を定める．

表 1: 各変数と精度

変数	名称	精度 [bit]
w	パラメータ	b_w
x	学習データの特徴量	b_x
t	学習データの正解ラベル	b_y

3.2 除算や指数関数の効率的な処理

秘密計算はソフトマックス関数の計算に必要な除算や指数関数の処理コストが大きい．そこで本論文では，秘密一括写像 (map)[32] を用いてこれらを効率的に処理する．秘密一括写像 (map) はルックアップテーブルを計算する関数で，定義域と値域を任意に定めることができる．

また，秘密一括写像は比較的大きなサイズの入力ベクトルに対して要素ごとの写像を計算することを目的としており，大きな入力に対しても高速に処理できる．1 回の秘密一括写像は，定義域や値域のサイズに比例するオーバーヘッドがかかるため，10^4 程度のサイズの写像を用意するのが現実的である．

秘密一括写像を用いることで，ソフトマックス関数などの除算や指数関数を含む活性化関数を効率良く計算できるだけでなく，これまで秘密計算での適用が難しかった Adam などの最適化手法も含めたアルゴリズムが実現可能である．

3.3 秘密計算 back prop アルゴリズム

単純な勾配降下法と Adam で共通する部分，すなわち式 (2.10) までのバックプロパゲーションの計算を秘密に行う，秘密計算 back prop アルゴリズムを表 2 に示す．隠れ層の数は n である．

3.3.1 秘密一括写像によるソフトマックス関数

表 2 中の softmax は，$f(x) = e^x$ を計算する map と，$f(x) = \frac{1}{x}$ を計算する map の 2 つを用いて実現する．具体的には，式 (3.1) のように，式 (2.3) の分母・分子をともに $\exp(u_i)$ で割ったものを考える．$j = 1, 2, \cdots, k$ に対して $u_j - u_i$ を計算してから，$f(x) = e^x$ を計算する map へ入力し，その結果を全て加算すると $\sum_{j=1}^{k}(\exp(u_j - u_i))$ が計算できる．加算結果を $f(x) = \frac{1}{x}$ を計算する map に入力することで，softmax の結果を得る．

$$\mathsf{softmax}(u_i) = \frac{1}{\sum_{j=1}^{k}(\exp(u_j - u_i))} \quad (3.1)$$

3.3.2 最適化を用いないパラメータ更新アルゴリズム

最適化を適用しない場合は式 (2.11) でパラメータを更新する．その際，学習率とバッチサイズの比 $\frac{\eta}{m}$ との乗算を効率的に計算するために，式 (3.2) のように，$\frac{\eta}{m}$ を 2 べきの数 2^{-H} で近似する．そのため，η, m はともに 2 べきの値にしておくのが良い．

$$H = -\lfloor \log_2 \frac{\eta}{m} \rfloor \quad (3.2)$$

最適化を適用しない場合のパラメータ更新アルゴリズムを表 3 に示す．表 2 と表 3 を順に任意の学習回数分繰り返すことでパラメータの学習を行う．ただし，パラメータの初期化は 1 回目の学習の時のみ行う．

3.3.3 Adam を用いたパラメータ更新アルゴリズム

Adam を適用した秘密計算ニューラルネットワークアルゴリズムでは表 1 に加えて，表 4 の精度を定めてお

表 2: 秘密計算 back prop アルゴリズム

入力 1：学習データの特徴量 $[\![X]\!]$
入力 2：学習データの正解ラベル $[\![T]\!]$
パラメータ：l 層と $l+1$ 層間のパラメータ $[\![W^l]\!]$
出力：l 層と $l+1$ 層間の勾配 $[\![G^l]\!]$

	計算式	精度 [bit]
1:	全ての $[\![W]\!]$ を He の初期値で初期化	b_w
2:	① 順伝播の計算	-
3:	$[\![U^1]\!] \leftarrow [\![W^0]\!] \cdot [\![X]\!]$	$b_w + b_x$
4:	$[\![Y^1]\!] \leftarrow \mathsf{ReLU}([\![U^1]\!])$	$b_w + b_x$
5:	**for** $i = 1$ **to** $n-1$ **do**	-
6:	$[\![U^{i+1}]\!] \leftarrow [\![W^i]\!] \cdot [\![Y^i]\!]$	$2b_w + b_x$
7:	$[\![Y^{i+1}]\!] \leftarrow \mathsf{ReLU}([\![U^{i+1}]\!])$	$2b_w + b_x$
8:	$[\![Y^{i+1}]\!] \leftarrow \mathsf{rshift}([\![Y^{i+1}]\!], b_w)$	$b_w + b_x$
9:	**end for**	-
10:	$[\![U^{n+1}]\!] \leftarrow [\![W^n]\!] \cdot [\![Y^n]\!]$	$2b_w + b_x$
11:	$[\![Y^{n+1}]\!] \leftarrow \mathsf{softmax}([\![U^{n+1}]\!])$	b_y
12:	② 逆伝播の計算	-
13:	$[\![Z^{n+1}]\!] \leftarrow [\![Y^{n+1}]\!] - [\![T]\!]$	b_y
14:	$[\![Z^n]\!] \leftarrow \mathsf{ReLU}'([\![U^n]\!]) \circ ([\![Z^{n+1}]\!] \cdot [\![W^n]\!])$	$b_w + b_y$
15:	$[\![Z^n]\!] \leftarrow \mathsf{rshift}([\![Z^n]\!], b_y)$	b_w
16:	**for** $i = 1$ **to** $n-1$ **do**	-
17:	$[\![Z^{n-i}]\!] \leftarrow \mathsf{ReLU}'([\![U^{n-i}]\!]) \circ ([\![Z^{n-i+1}]\!] \cdot [\![W^{n-i}]\!])$	$2b_w$
18:	$[\![Z^{n-i}]\!] \leftarrow \mathsf{rshift}([\![Z^{n-i}]\!], b_w)$	b_w
19:	**end for**	-
20:	③ 勾配の計算	-
21:	$[\![G^0]\!] \leftarrow [\![Z^1]\!] \cdot [\![X]\!]$	$b_w + b_x$
22:	**for** $i = 1$ **to** $n-1$ **do**	-
23:	$[\![G^i]\!] \leftarrow [\![Z^{i+1}]\!] \cdot [\![Y^i]\!]$	$2b_w + b_x$
24:	**end for**	-
25:	$[\![G^n]\!] \leftarrow [\![Z^{n+1}]\!] \cdot [\![Y^n]\!]$	$b_w + b_x + b_y$

表 3: 最適化を用いないパラメータ更新アルゴリズム

入力：l 層と $l+1$ 層間の勾配 $[\![G^l]\!]$
出力：更新した l 層と $l+1$ 層間のパラメータ $[\![W^l]\!]$

	計算式	精度 [bit]
1:	$[\![G^0]\!] \leftarrow \mathsf{rshift}([\![G^0]\!], b_x + H)$	b_w
2:	$[\![W^0]\!] \leftarrow [\![W^0]\!] - [\![G^0]\!]$	b_w
3:	**for** $i = 1$ **to** $n-1$ **do**	-
4:	$[\![G^i]\!] \leftarrow \mathsf{rshift}([\![G^i]\!], b_w + b_x + H)$	b_w
5:	$[\![W^i]\!] \leftarrow [\![W^i]\!] - [\![G^i]\!]$	b_w
6:	**end for**	-
7:	$[\![G^n]\!] \leftarrow \mathsf{rshift}([\![G^n]\!], b_x + b_y + H)$	b_w
8:	$[\![W^n]\!] \leftarrow [\![W^n]\!] - [\![G^n]\!]$	b_w

く必要がある．これ以降，$\frac{1}{1-\beta_1^t} = \hat{\beta}_{1,t}$，$\frac{1}{1-\beta_2^t} = \hat{\beta}_{2,t}$，$\frac{\eta}{\sqrt{\hat{v}}+\epsilon} = \hat{g}$ と表記する．$\hat{\beta}_{1,t}$ および $\hat{\beta}_{2,t}$ は事前に各 t に対して計算しておき，$\hat{g}$ の計算は $\hat{v}$ を入力として，$\frac{\eta}{\sqrt{\hat{v}}+\epsilon}$ を出力する map を用いて実現する．そして，その map を $\mathsf{Adam}(\hat{v})$ と表記する．

また，Adam を用いたパラメータ更新アルゴリズムでは，バッチサイズ m での割算を rshift で処理する．そのため，バッチサイズ m は 2 べきの値にしておくのが良く，その際のシフト量 H' を式 (3.3) で求める．

$$H' = -\lfloor \log_2 m \rfloor \tag{3.3}$$

Adam を適用した場合のパラメータ更新アルゴリズムを表 5 に示す．表 2 と表 5 を順に任意の学習回数分繰り返すことでパラメータの学習を行う．ただし，パラメータの初期化は 1 回目の学習の時のみ行う．

表 4: 各値と精度

値	精度 [bit]
β_1, β_2	b_β
$\hat{\beta}_{1,t}$	$b_{\hat{\beta}_1}$
$\hat{\beta}_{2,t}$	$b_{\hat{\beta}_2}$
$\hat{g}$	$b_{\hat{g}}$

表 5: Adam を用いたパラメータ更新アルゴリズム

入力 1：l 層と $l+1$ 層間の勾配 $[\![G^l]\!]$
入力 2：学習回数 t
出力：更新した l 層と $l+1$ 層間のパラメータ $[\![W^l]\!]$

	計算式	精度 [bit]
1:	$[\![G^0]\!] \leftarrow \mathsf{rshift}([\![G^0]\!], b_x + H')$	b_w
2:	**for** $i = 1$ **to** $n-1$ **do**	-
3:	$[\![G^i]\!] \leftarrow \mathsf{rshift}([\![G^i]\!], b_w + b_x + H')$	b_w
4:	**end for**	-
5:	$[\![G^n]\!] \leftarrow \mathsf{rshift}([\![M^i]\!], b_x + b_y + H')$	b_w
6:	**for** $i = 0$ **to** n **do**	-
7:	$[\![M^i]\!] \leftarrow \beta_1 [\![M^i]\!] + (1-\beta_1)[\![G^i]\!]$	$b_w + b_\beta$
8:	$[\![M^i]\!] \leftarrow \mathsf{rshift}([\![M^i]\!], b_\beta)$	b_w
9:	$[\![V^i]\!] \leftarrow \beta_2 [\![V^i]\!] + (1-\beta_2)[\![G^i]\!] \circ [\![G^i]\!]$	$2b_w + b_\beta$
10:	$[\![V^i]\!] \leftarrow \mathsf{rshift}([\![V^i]\!], b_\beta)$	$2b_w$
11:	$[\![\hat{M}^i]\!] \leftarrow \hat{\beta}_{1,t} [\![M^i]\!]$	$b_w + b_{\hat{\beta}_1}$
12:	$[\![\hat{V}^i]\!] \leftarrow \hat{\beta}_{2,t} [\![V^i]\!]$	$2b_w + b_{\hat{\beta}_2}$
13:	$[\![\hat{G}^i]\!] \leftarrow \mathsf{Adam}([\![\hat{V}^i]\!])$	$b_{\hat{g}}$
14:	$[\![\hat{G}^i]\!] \leftarrow [\![\hat{G}^i]\!] \circ [\![\hat{M}^i]\!]$	$b_{\hat{g}} + b_w + b_{\hat{\beta}_1}$
15:	$[\![\hat{G}^i]\!] \leftarrow \mathsf{rshift}([\![\hat{G}^i]\!], b_{\hat{g}} + b_{\hat{\beta}_1})$	b_w
16:	$[\![W^i]\!] \leftarrow [\![W^i]\!] - [\![\hat{G}^i]\!]$	b_w
17:	**end for**	-

4 実験

4.1 実験設定

表 6 に示すマシン 3 台を用いて実験を行った．

表 6: 測定環境

OS	CentOS Linux release 7.3.1611
CPU	Intel Xeon Gold 6144k(3.50GHz 8 コア/16 スレッド) × 2
メモリ	768GB
NW	Intel Ethernet Controller X710/X557-AT 10G リング構成

モデル設定： 隠れ層の数 $n = 2$，各隠れ層のニューロン数 $d_1, d_2 = 128$ というディープニューラルネットを用いて実験を行った．各実験において，設定が明記されていない部分に関しては，すべて同じ設定を用いている．

精度設定： $b_w = 20$[bit]，$b_x = 8$[bit]，$b_y = 14$[bit]，$b_\beta = 10$[bit]，$b_{\hat{\beta}_1} = 17$[bit]，$b_{\hat{\beta}_2} = 4$[bit]，$b_{\hat{g}} = 10$[bit]

ハイパーパラメータ： 全ての実験でバッチサイズ $m = 128$ である．最適化を用いない実験では，SecureML と同様に学習率を $\eta = 2^{-7}$ とした．Adam のハイパーパラメータは全て提案論文 [18] で推奨されている $\beta_1 = 0.9$，$\beta_2 = 0.999$，$\eta = 0.001$，$\epsilon = 10^{-8}$ とした．

その他の設定： パラメータは全て He の初期値に従って初期化している．事前に学習データに対して標準化等の処理は加えず，今回の設定ではバイアス項を考慮しない．また，1epoch ごとにシャッフルを行っている．

4.2 MNIST データでの実験

MNIST データセット [7] を用い，SecureML[21] や SecureNN [28] と同じ設定で実験を行った．MNIST データセットは 0~9 の手書き文字を識別するためのデータセットで，28×28 の白黒画像データが学習用として 6 万件あり，それぞれの画像に対応した 0~9 の値が入った正解ラベルも 6 万件ある．そして学習結果のテスト用として，同じ形式の画像データと正解ラベルが 1 万件ずつ用意されている．したがって，MNIST データセットを学習・予測する場合，入力層のニューロン数 $d_0 = 784$，出力層のニューロン数 $d_3 = 10$ となる．

4.2.1 処理時間と予測精度

最適化を用いずソフトマックス関数のみ実装した場合を提案手法 1(SGD)，Adam を適用したものを提案手法 2(Adam) として，実験結果を表 7 に示す．予測精度はテスト用データを用いて評価している．比較対象とした SecureML および SecureNN は，提案手法 1(SGD) と同じ学習アルゴリズムを用いており，ソフトマックス関数の計算において指数関数を ReLU 関数で近似している点が，本論文と異なる．モデルの設定やバッチサイズの設定は同一である．

表 7: 処理時間と予測精度

	Epoch 数	時間 [sec]	予測精度 [%]
SecureML	15	25283	93.4
SecureNN	15	3708	93.4
平文※ (SGD)	15	40	93.94
提案手法 1 (SGD)	15	3736	93.84
平文※ (Adam)	15	12	95.54
提案手法 2 (Adam)	1	412	94.2

※平文は scikit-learn で実装したもので，測定環境は Intel core-i7 8700 3.2GHz× 3 コア/メモリ 8GB/CentOS. データ読み込みに 10 秒程度かかっており，SGD でも Adam でも 1epoch あたりの処理時間は 2 秒程度であった．

提案手法 1(SGD) の考察 表 7 から，近似ソフトマックス関数で計算した先行研究よりも，提案手法 1(SGD) は平文の予測精度に近い結果が得られることが分かった．ここで出た誤差は，初期値を平文と異なる乱数で初期化していること，平文とはシャッフルに使用した乱数が異なっていたことが要因だと考えられる．今後の課題として，初期値やシャッフル等の設定も揃えて，より精密に平文との比較を行う必要がある．また，先行研究と同様の近似ソフトマックス関数を用いた場合は，初期化処理＋1epoch 分の処理時間が 207 秒程度であり，15epoch 行った場合は 3000 秒程度になることが見込まれる．これは SecureNN の結果よりも 2 割程度高速であり，ベースとなるライブラリの性能は MEVAL がやや高かったと考えられる．

提案手法 2(Adam) の考察 表 7 から，提案手法 2(Adam) では 1epoch で先行研究よりも高い予測精度が得られた．平文の結果と比較して予測精度に差異が見られたのは，提案手法 1(SGD) と同じ要因に加えて，秘密一括写像が増えたことによって数値誤差が大きくなったことが考えられる．実際に，$b_{\hat{g}}$ の精度を上げることによって予測精度の向上が見られた．秘密一括写像の精度と処理時間はトレードオフであるため，今後の課題として様々な精度での実験を行い，最適な精度を検討する必要がある．

4.2.2 バッチサイズを変えた実験

提案手法 2(Adam) を用いた設定で，バッチサイズのみを変化させて 1epoch 分の処理時間を測定した結果を表 8 に示す．

表 8: 各バッチサイズの処理時間

バッチサイズ	処理時間 [s]
128	412
256	215
512	122

測定値から，処理時間がバッチサイズにほぼ反比例，すなわち性能がバッチサイズに比例していることが分かる．秘密分散ベースの秘密計算には，通信量の他，通信回数がかかりネットワーク遅延がボトルネックになることがある．バッチサイズがこの程度の小さい値の場合には通信量よりも遅延がボトルネックとなり，1 バッチごとの処理時間がバッチサイズに非依存となっていると考えられる．バッチサイズは平文上ではキャッシュに載るかどうかなど性能観点と，収束の挙動で決められている．この実験結果は，秘密計算において平文上とは異なるバッチサイズを探求する意義を示しているといえる．

5 おわりに

本論文では秘密計算上のディープラーニングにおいて以下の 2 つを初めて実現し，高速かつ高精度な実装を報告した．

- 出力層の活性化関数であるソフトマックス関数
- 最適化手法 Adam

固定小数点数と右シフトの組み合わせ，および秘密一括写像によるソフトマックス関数の計算によって，処理コストの大きい浮動小数点数や除算などを用いずに，効率良く秘密計算でバックプロパゲーションを処理可能な秘密計算 back prop アルゴリズムを実現した．提案手法では既存研究における ReLU 関数を用いた関数とは異なり，ソフトマックス関数を理論的に近似できているため，既存研究では MNIST データセットにおける個別の精度の提示であったのに対して，提案手法ではどのようなデータセットでも平文上と同等の精度を保証する．

また，単純な最急降下法よりも収束スピードが非常に速い最適化手法である Adam を秘密一括写像を用いて実現し，複雑な処理が入って性能が落ちる要因よりも，収束が高速になる要因の方が提案手法と本論文の実装の場合大きいことを示した．結果的には既存研究の約 9 倍，また既存研究と同じ最急降下法を用いた提案手法 1 と比べてもほぼ同じ約 9 倍高速化されたことを報告した．

5.1 今後の課題

本論文の実験で平文の SGD と提案手法で予測精度 0.1%と，想定より大きな差となった．これはシャッフルに用いた乱数が異なることで発生していると考えられ，統一した乱数による検証を行う，また，1000 万件等の大規模データへの適用性を検証していく．

参考文献

[1] caffe. http://caffe.berkeleyvision.org/.

[2] deep genomics. https://www.deepgenomics.com/.

[3] Eigen library. http://eigen.tuxfamily.org/.

[4] Enlitic. https://www.enlitic.com/.

[5] keras. https://keras.io/ja/optimizers/.

[6] lasagne. https://lasagne.readthedocs.io/en/latest/.

[7] Mnist database. http://yann.lecun.com/exdb/mnist/.

[8] scikit-learn. https://scikit-learn.org/stable/.

[9] Trusting social. https://trustingsocial.com/.

[10] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *CCS*, pp. 805–817, 2016.

[11] Assi Barak, Daniel Escudero, Anders Dalskov, and Marcel Keller. Secure evaluation of quantized neural networks.

[12] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *TCC*, pp. 342–362, 2005.

[13] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[16] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing schemes realizing general access structure. In *Proc. of the IEEE Global Telecommunication Conf., Globecom 87*, pp. 99–102, 1987. Journal version: Multiple assignment scheme for sharing secret. J. of Cryptology, 6(1):15-20, 1993.

[17] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1651–1669, 2018.

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

[19] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

[20] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 619–631. ACM, 2017.

[21] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy, SP 2017*, pp. 19–38, 2017.

[22] PwC コンサルティング合同会社. 日本企業における ai 活用の可能性 -成功のカギはどこにあるのか. https://www.pwc.com/jp/ja/knowledge/thoughtleadership/2018/assets/pdf/tmt1801.pdf.

[23] Michael O. Rabin. How to exchange secrets by oblivious transfer. In *Technical Report TR-81*, 1981.

[24] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[25] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[26] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, Vol. 5, No. 3, p. 1, 1988.

[27] Adi Shamir. How to share a secret. *Communications of the ACM*, Vol. 22, No. 11, pp. 612–613, 1979.

[28] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, Vol. 1, p. 24, 2019.

[29] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pp. 162–167, 1986.

[30] 三品気吹, 濱田浩気, 菊池亮, 五十嵐大. 秘密計算によるロジスティック回帰は本当に使えるか？ In *SCIS*, 2018.

[31] 三品気吹, 五十嵐大, 濱田浩気, 菊池亮. 高精度かつ高効率な秘密ロジスティック回帰の設計と実装. In *CSS*, 2018.

[32] 濱田浩気, 五十嵐大, 千田浩司. 秘匿計算上の一括写像アルゴリズム. 電子情報通信学会論文誌 A, Vol. 96, No. 4, pp. 157–165, 2013.

[33] 総務省. 平成 29 年版 情報通信白書 第 1 部. http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h29/pdf/n2200000.pdf.

[34] 五十嵐大, 濱田浩気, 菊池亮, 千田浩司. 超高速秘密計算ソートの設計と実装：秘密計算がスクリプト言語に並んだ日. In *CSS*, 2017.

[35] 桐淵直人, 五十嵐大, 濱田浩気, 菊池亮. プログラマブルな秘密計算ライブラリ MEVAL3. *SCIS*, 2018.

[36] 日刊工業新聞. Ai 研究、次に稼げる分野は材料 !? https://newswitch.jp/p/11548.

[37] 日本製薬工業協会. 製薬企業（国内）における人工知能（ai）の利用状況及び利用への取り組みに関するアンケート. http://www.jpma.or.jp/medicine/shinyaku/tiken/allotment/pdf/use-ai.pdf.