実用的な秘密計算ディープラーニングの実現

三品 気吹[1,a)]　濱田 浩気[1]　五十嵐 大[1]

概要：本稿では，データを暗号化したまま処理しながらも実用的な性能を持つ，秘密計算ディープラーニングの実現を目指す．ディープラーニングは最も有望な AI 手法で，難しい分析や予測が可能になると期待される一方，AI 分析で個人データ等を用いることに対して，使う側・使われる側ともに不安やリスクも感じている．秘密計算ディープラーニングは，こうした不安やリスクを秘密計算によって低減することで，これまでより活発な AI 利活用の足掛かりになると考えられる．しかし従来の秘密計算ディープラーニングでは (1) 処理速度 (2) モデル・分析の自由さ (3) 大規模データでの実用性といった点で課題を抱えている．本稿ではそれらの課題に取り組み (1)784 属性 ×6 万件を 5 分で処理，予測精度 95.64%を達成 (2) 既存より深いネットワーク・回帰分析への対応 (3)100 属性 ×1000 万件を 2 時間で処理，という結果を示した．

キーワード：秘密計算，機械学習，ニューラルネットワーク，ディープラーニング

Realization of Practical Secure Deep Learning

Ibuki Mishina[1,a)]　Koki Hamada[1]　Dai Ikarashi[1]

Abstract: In this paper, we aim at the realization of practical secure deep learning that can process data with encryption. Deep learning is the most promising AI method, and it is expected to enable difficult analysis and prediction. On the other hand, both data users and data owners feel anxiety and risk about using personal data in AI analysis. Secure deep learning provides a foothold for more active use of AI analysis by reducing such anxiety and risk through secure computation. However, there are 3 issues with secure deep learning in previous research, (1) Processing speed (2) Ability to select model or/and analysis methods (3) Practicality when learning with large-scale data. In this paper, we addressed these issues and presented the following results. (1) Processing 784×60,000 in 5 minutes, achieving a prediction accuracy of 95.64 % (2) Realization of deeper networks and regression analysis (3) Processing 100×10 million in 2 hours,

Keywords: Secure Computation, Machine Learning, Neural Network, Deep Learning

1. はじめに

秘密計算は，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，企業の重要な情報や顧客情報を安全に守ったまま，データ分析などに利用できる．その分析手法として特に有望なのが機械学習 (AI) である．筆者らは，平文の AI ライブラリ [6] が提供するような数多の AI 分析を，データを安全に守ったまま行う「秘密計算 AI ライブラリ」の実現を目指している．それに向け，筆者らはこれまでに秘密計算上のロジスティック回帰分析に取り組んでおり，R と比較しても遜色無い処理速度・予測精度を達成した [22], [23]．また AI 手法の中でも，特に様々領域で利活用が期待されている，ディープラーニングを秘密計算上で実現し，先行研究より処理速度・予測精度の両方で高い性能を示した [21]．本稿では実用的な秘密計算ディープラーニング実現に向け，更なる性能向上を目指す．

1.1 関連研究

秘密計算上でディープラーニングを実装した先行研究には，SecureML[14]，ABY^3[13]，SecureNN[19] がある．Se-

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories
[a)] ibuki.mishina.br@hco.ntt.co.jp

cureML は秘密分散，garbled circuit[20]，紛失通信 [15] などを組み合わせ，学習と予測を実装している．MNIST[5] を学習データに用いた実験では処理時間が約 7 時間，予測精度は 93.4%である．ABY^3 は新たなフレームワーク (ABY^3) を実装し，それを用いた学習と予測を実装している．SecureML と同じ設定の実験では処理時間 45 分，予測精度 94%である．SecureNN は加法秘密分散を用いて学習と予測を実装している．SecureML と同じ設定の実験では処理時間約 1 時間，予測精度は 93.4%である．

筆者らも秘密分散を用いたディープラーニングを実装している [21]．秘密一括写像によって，ソフトマックス関数の高精度な計算と，Adam という収束の速い学習アルゴリズムを初めて秘密計算上で実現し，SecureML と同じ設定の実験では処理時間約 7 分，予測精度 94.2%を達成した．

1.2 先行研究の課題

1.2.1 処理速度

平文の AI ライブラリ [6] を用いて，筆者らの先行研究 [21] と同じ設定・学習アルゴリズムとした場合，処理時間は約 18 秒となった．[21] は既に他者の先行研究より 6〜7 倍高速だが，まだ改善の余地があると考えられる．

1.2.2 モデル・分析の自由さ

ニューラルネットワークはモデル設計の自由さが特徴であり，平文の深層学習ライブラリ [3], [4] では隠れ層の数 (深さ) やニューロン数を自由に設定できるが，先行研究では殆ど決まった形のモデルで実験されており，より深いネットワーク等での実用性は確認されていない．また，出力層の活性化関数を変えることで，クラス分類だけでなく回帰分析もできるが，その両方を実現した先行研究は無い．

1.2.3 大規模データでの実用性

ディープラーニングでは，大規模な学習データを用いることも多い．しかし先行研究で用いられている MNIST[5] のデータ数は 6 万件とあまり多くなく，大規模データを用いた場合の実用性は，これまで確認されていない．

1.3 本稿の成果

本稿では先述の課題に取り組み，以下の成果を達成した．

(1) 秘密計算ディープラーニングの更なる高速化

(2) より大きなネットワークへの対応，分析の拡充

(3) 大規模な学習データでの実用性の確認

1.4 秘密計算と AI

AI 利活用による業務効率化を目指す領域は，マーケティング，医療，金融など多岐に渡る．特にディープラーニングは他の AI 手法では難しい分析も可能とされ，レントゲンや MRI 画像による病気の診断 [2]，ソーシャルデータや電話代の支払い履歴による信用の判定 [7] などに利用されている．一方で，AI 分析で個人データ等を扱うことに対してリスクを感じる企業も多く，AI によるデータ利活用の障壁となっている．そこで，データ利活用に対する阻害要因の 1 つであるプライバシーに関するリスクを低減し，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，秘密計算を適用する．秘密計算のデータを暗号化したたまま計算できる性質と，AI を組み合わせることで，安全かつ活発なデータ利活用の足掛かりになると考えられる．

2. 準備

2.1 記法

a を b で定義することを $a := b$ と書き，ベクトルを $\vec{a} := (a_0, \ldots, a_{n-1})$ と書き，$i \times j$ の行列を $a_{i,j}$ と書き，$(a_{i,j})^T$ は $a_{i,j}$ の転置行列を表す．また，2 つの行列や同じ要素数の 2 つのベクトルの内積は $(\cdot)$ で表し，要素ごとの積は $(\circ)$ で表す．演算子が書かれていないものはスカラー倍である．Z_2 を $Z/2Z$ とする．$[\cdot]$ は述語を表し，例えば $[a > b]$ は $a > b$ ならば 1，そうでなければ 0 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種類があるが，本稿では図 1 に示すような全結合・順伝播型を用いる．隠れ層が 2 層以上のものをディープニューラルネットワークと呼び，それを学習することをディープラーニングと呼ぶ．

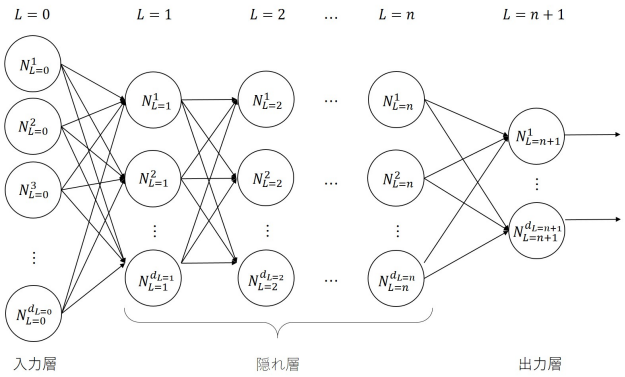

図 1 全結合・順伝搬型ニューラルネットワーク

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^i_j 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，各ニューロンの結合の強さをパラメータと呼び，適切な出力を与えるパラメータの推定が学習である．

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．

ReLU 関数

隠れ層が 2 層以上の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いる [9][16].

$$\mathsf{ReLU}(u) = \mathsf{max}(0, u) \tag{1}$$

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．

$$\mathsf{ReLU}'(u) = \begin{cases} 0(u \leqq 0) \\ 1(u > 0) \end{cases} \tag{2}$$

ソフトマックス関数

画像識別のようなクラス分類を解く場合，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式 (3) のようになる．e はネイピア数である．

$$\mathsf{softmax}(u_i) = \frac{e^{u_i}}{\sum_{j=0}^{k-1} e^{u_j}} \tag{3}$$

また出力層がソフトマックス関数の場合，損失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 $E(w)$ は式 (4) のようになる．

$$E(w) = -\sum_m \sum_k t_{m,k} \circ \log y_{m,k} \tag{4}$$

$E(w)$ の $u_{m,k}$ での偏微分は式 (5) のようになる．$u_{m,k}$ については 2.2.2 を参考にされたい．

$$\frac{\partial E(w)}{\partial u_{m,k}} = y_{m,k} - t_{m,k} \tag{5}$$

恒等関数

ニューラルネットワークにおいて株価予測のような回帰分析を行う場合，出力層には恒等関数 $f(x) = x$ を用いる．その場合の損失関数としては，式 (6) のような二乗誤差を利用する．$E(w)$ が二乗誤差の場合の偏微分も式 (5) と同様 $y - t$ の形になる．

$$E(w) = \sum_m \frac{(t_m - y_m)^2}{2} \tag{6}$$

2.2.2 バックプロパゲーション

ディープラーニングではバックプロパゲーションを用いる [18]．表 1 に，ミニバッチ確率的勾配降下法 [12] によるバックプロパゲーションの処理を示す．w^l は l 層目と $l+1$ 層目の間のパラメータ，n は隠れ層の数，m はバッチサイズを表し，全データを 1 回使いきることを 1epoch と表現する．ActH と ActH′ は任意の隠れ層の活性化関数とその微分，ActO は任意の出力層の活性化関数を表す．

表 1 バックプロパゲーション

入力 1：学習データの特徴量 x_{m,d_0}
入力 2：学習データのラベル $t_{m,d_{n+1}}$
入力 3：初期化したパラメータ $w^l(l = 0, \cdots, n)$
ハイパーパラメータ：学習率 η
出力：学習したパラメータ $w^l(l = 0, \cdots, n)$

① 順伝播の計算
$u^1_{m,d_1} \leftarrow x_{m,d_0} \cdot w^0_{d_0,d_1}$
$y^1_{m,d_1} \leftarrow \mathsf{ActH}(u^1_{m,d_1})$
for $l \leftarrow 1$ **to** $n-1$ **do**
  $u^{l+1}_{m,d_{l+1}} \leftarrow y^l_{m,d_l} \cdot w^l_{d_l,d_{l+1}}$
  $y^{l+1}_{m,d_1} \leftarrow \mathsf{ActH}(u^{l+1}_{m,d_{l+1}})$
end for
$u^{n+1}_{m,d_{n+1}} \leftarrow y^n_{m,d_n} \cdot w^n_{d_n,d_{n+1}}$
$y^{n+1}_{m,d_{n+1}} \leftarrow \mathsf{ActO}(u^{n+1}_{m,d_{n+1}})$
② 逆伝播の計算
$z^{n+1}_{m,d_{n+1}} \leftarrow y^{n+1}_{m,d_{n+1}} - t_{m,d_{n+1}}$
for $l \leftarrow 0$ **to** $n-1$ **do**
  $z^{n-l}_{m,d_{n-l}} \leftarrow \mathsf{ActH}'(u^{n-l}_{m,d_{n-l}}) \circ (z^{n-l+1}_{m,d_{n-l+1}} \cdot (w^{n-l}_{d_{n-l},d_{n-l+1}})^T)$
end for
for $l \leftarrow 0$ **to** n **do**
  ③ 勾配の計算
  $g^l_{d_l,d_{l+1}} = (y^l_{m,d_l})^T \cdot z^{l+1}_{m,d_{l+1}}$
  ④ パラメータの更新
  $w^l_{d_l,d_{l+1}} = w^l_{d_l,d_{l+1}} - \frac{\eta}{m} g^l_{d_l,d_{l+1}}$
end for

2.2.3 学習アルゴリズムの最適化:Adam

表 1 は単純な勾配降下法によってパラメータを更新する最も基本的な学習アルゴリズムである．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あり，それらを解決するための最適化手法が [17] で紹介されている．最適化の適用によって先述の問題を防げるため，平文の AI ライブラリでも提供されている [3], [6]．本稿では，[17] で特に良いとされていた Adam を用いる．Adam を適用する場合でも，表 1 の勾配の計算までは全く同じ処理で，パラメータの更新のみが異なる．

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，例えば 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り，m, v は 0 で初期化している．ここでの上付き数字 t は t 乗を表す．Adam の提案論文 [11] では $\beta_1 = 0.9$，$\beta_2 = 0.999$，学習率 $\eta = 0.001$，$\epsilon = 10^{-8}$ となっている．ϵ は 0 での除算を防ぐための値である．

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1) g_t \tag{7}$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2) g_t \circ g_t \tag{8}$$

$$\hat{m}_{t+1} = \frac{1}{1 - \beta_1^t} m_{t+1} \tag{9}$$

$$\hat{v}_{t+1} = \frac{1}{1 - \beta_2^t} v_{t+1} \tag{10}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_{t+1}} + \epsilon} \circ \hat{m}_{t+1} \tag{11}$$

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方式があり，中でも秘密情報を「シェア」という複数の断片に変換する秘密分散方式は，データの処理単位が小さく，処理が高速である [8], [25]．本稿では，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方式を用いる．

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 [26]．MEVAL は演算に応じて複数の秘密分散を用いているが，本稿では主に $\mathrm{GF}(2^{61}-1)$ 上の Shamir 秘密分散と Z_2 上の複製秘密分散を用い，それぞれのシェアを $[\![\vec{a}]\!], \{\vec{a}\}$ と書く．

シェアの定数倍や加算は通信無しで実行でき，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 が持つ演算のうち，本稿で特に重要なものを以下に示す．処理単位はベクトルである．

- $\mathsf{rshift}([\![\vec{a}]\!], t)$:
  - 入力：$[\![\vec{a}]\!] := ([\![a_0]\!], \ldots, [\![a_{m-1}]\!])$ と t
  - 出力：$[\![\vec{a}]\!]$ の各要素を t[bit] 算術右シフトした $[\![\vec{b}]\!] := ([\![b_0]\!], \ldots, [\![b_{m-1}]\!])$

  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 [23] 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．

$$[\![A' \times 2^n]\!] = [\![A \times 2^n]\!] + a \times 2^n (a \geq |A|) \quad (12)$$

$$[\![A' \times 2^m]\!] = \mathsf{rlshift}([\![A' \times 2^n]\!], n-m) \quad (13)$$

$$[\![A \times 2^m]\!] = [\![A' \times 2^m]\!] - a \times 2^m \quad (14)$$

- $\mathsf{greater\_than}([\![\vec{a}]\!], t)$:
  - 入力：$[\![\vec{a}]\!] := ([\![a_0]\!], \ldots, [\![a_{m-1}]\!])$ と t
  - 出力：$\{[\vec{a} > t]\} = (\{[a_0 > t]\}, \ldots, \{[a_{m-1} > t]\})$
- $\mathsf{toP}(\{\vec{a}\})$:
  - 入力：$\{\vec{a}\} := (\{a_0\}, \ldots, \{a_{m-1}\})$
  - 出力：環を変換した $[\![\vec{a}]\!] := ([\![a_0]\!], \ldots, [\![a_{m-1}]\!])$
- $\mathsf{sum}([\![\vec{a}]\!])$:
  - 入力：$[\![\vec{a}]\!] := ([\![a_0]\!], \ldots, [\![a_{m-1}]\!])$
  - 出力：$[\![\sum_{i=0}^{m-1} a_i]\!]$
- $\mathsf{exp}([\![\vec{a}]\!], b_\alpha, b_\beta)$:
  - 入力：$[\![\vec{a}]\!] := ([\![a_0]\!], \ldots, [\![a_{m-1}]\!])$ (小数部分のビット数が b_α[bit])
  - 出力：$[\![e^{\vec{a}}]\!] := ([\![e^{a_0}, \ldots, e^{a_{m-1}}]\!])$ (小数部分のビット数が b_β[bit])
- $\mathsf{reciprocal}([\![\vec{a}]\!], b_\alpha, b_\beta)$:
  - 入力：$[\![\vec{a}]\!] := ([\![a_0]\!], \ldots, [\![a_{m-1}]\!])$ (小数部分のビット数が b_α[bit])
  - 出力：$[\![\frac{\vec{1}}{\vec{a}}]\!] := ([\![\frac{1}{a_0}, \ldots, \frac{1}{a_{m-1}}]\!])$ (小数部分のビット数が b_β[bit])
- $\mathsf{sqrtinv}([\![\vec{a}]\!], b_\alpha, b_\beta)$:
  - 入力：$[\![\vec{a}]\!] := ([\![a_0]\!], \ldots, [\![a_{m-1}]\!])$ (小数部分のビット数が b_α[bit])
  - 出力：$[\![\frac{\vec{1}}{\sqrt{a}}]\!] := ([\![\frac{1}{\sqrt{a_0}}, \ldots, \frac{1}{\sqrt{a_{m-1}}}]\!])$ (小数部分のビット数が b_β[bit])

3. 提案手法

3.1 秘密計算上の除算・指数計算

筆者らが以前実装したディープラーニング [21] では，ソフトマックス関数の指数や除算，Adam の式 (11) を MEVAL の関数である秘密一括写像 [24] によって処理していた．秘密一括写像は任意の関数を任意の精度で計算できるため，秘密計算が苦手とする指数や除算などを効率良く処理する方法として非常に有効である．しかし，他の秘密計算が得意な処理 (e.g. 加算や乗算) と比較すると，どうしてもコストは大きく，[21] でもボトルネックとなっていた．そこで本稿では，各々の専用の関数 exp, reciprocal, sqrtinv を新たに導入し，更なる性能向上を目指す．

3.2 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，本稿では固定小数点数を用いてアルゴリズム設計を行う．変数や定数ごとに必要な精度が異なるため各々で設定しており，精度が α[bit] の場合，秘密情報 $[\![x]\!]$ は固定小数点数 $[\![x \times 2^\alpha]\!]$ として秘密分散される．

また固定小数点数で処理を行う場合，乗算によって精度が変わることが問題となる．バックプロパゲーションでは層ごとに乗算を行うため，そのままでは途中でオーバーフローしてしまう．そこで本稿では，rshift[23] を用いて意図的に桁を落とすことでオーバーフローを防ぎつつ，固定小数点数による高速なバックプロパゲーションを実現する．本稿では表 2 のように精度を定める．

表 2 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	学習データのラベル	b_y

3.3 Adam 適用 Secure-Back Prop アルゴリズム

Adam を適用したバックプロパゲーションを秘密計算で行う，Adam 適用 Secure-Back Prop アルゴリズムを表 4 に示す．隠れ層の数は n である．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く．また，Adam を適用する場合は表 3 のように各値の精度を設定しておく必要がある．

表 3 Adam で用いる各値の精度

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

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 での除算を 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 (15) で求める．

$$H = -\lfloor \log_2 m \rfloor \tag{15}$$

同様に学習率 η も 2 べきの値にするのが良い．たとえば Adam の提案論文では $\eta = 0.001$ が推奨されているため，$0.001 \fallingdotseq 2^{-10}$ という近似をし，学習率 η との乗算を，$\eta' = 10$[bit] の右シフトで処理すると効率が良い．

3.3.1 活性化関数の処理

[21] で提案したアルゴリズムでは，隠れ層と出力層の活性化関数が，それぞれ $\mathsf{ReLU}, \mathsf{softmax}$ のみに対応していたが，ニューラルネットワークでは活性化関数が自由に設定できるべきなので，本稿の提案アルゴリズム (表 4) では，任意の活性化関数で置き換え可能な設計にした．

隠れ層の活性化関数

表 4 中の ActH は，任意の隠れ層の活性化関数での処理に対応する．たとえば隠れ層の活性化関数が ReLU 関数だった場合，表 4 手順 3 の $[\![Y^1]\!] \leftarrow \mathsf{ActH}([\![U^1]\!], b_x)$ は，以下のように処理する．

$$[\![Y^1]\!] \leftarrow \mathsf{ReLU}([\![U^1]\!]) \tag{16}$$

$$[\![Y^1]\!] \leftarrow \mathsf{rshift}([\![Y^1]\!], b_x) \tag{17}$$

また，$[\![\vec{c}]\!] \leftarrow \mathsf{ReLU}([\![\vec{a}]\!])$ は MEVAL の関数を用いて以下のように計算する．この際，計算途中で得られる $[\![\vec{b}]\!]$ は，逆伝搬の計算時に用いる $\mathsf{ReLU}'([\![\vec{a}]\!])$ に相当する．

$$\{\vec{b}\} \leftarrow \mathsf{greater\_than}([\![\vec{a}]\!], 0) \tag{18}$$

$$[\![\vec{b}]\!] \leftarrow \mathsf{toP}(\{\vec{b}\}) \tag{19}$$

$$[\![\vec{c}]\!] \leftarrow [\![\vec{a}]\!] \circ [\![\vec{b}]\!] \tag{20}$$

表 4 手順 12 は ReLU' の場合，以下のように計算する．

$$[\![Z^n]\!] \leftarrow \mathsf{ReLU}'([\![U^n]\!]) \circ ([\![Z^{n+1}]\!] \cdot [\![W^n]\!]) \tag{21}$$

$$[\![Z^n]\!] \leftarrow \mathsf{rshift}([\![Z^n]\!], b_y) \tag{22}$$

ActH としてシグモイド関数や双曲線正接関数などを用いる場合は，MEVAL の関数である秘密一括写像 [24] や，$\mathsf{exp}, \mathsf{reciprocal}$ によって処理できる．筆者らは以前，秘密計算上でのシグモイド関数の処理方法を提案している [23]．

表 4 Adam 適用 Secure-Back Prop アルゴリズム

入力 1：学習データの特徴量 $[\![X]\!]$
入力 2：学習データのラベル $[\![T]\!]$
入力 3：初期化したパラメータ $[\![W^l]\!](l = 0, \cdots, n)$
入力 4：0 で初期化した $[\![M^l]\!], [\![V^l]\!](l = 0, \cdots, n)$
ハイパーパラメータ：$\eta', \beta_1, \beta_2, \hat{\beta}_{1,t}, \hat{\beta}_{2,t}$
出力：更新したパラメータ $[\![W^l]\!]$

	計算式	精度 [bit]
1:	① 順伝播の計算	-
2:	$[\![U^1]\!] \leftarrow [\![W^0]\!] \cdot [\![X]\!]$	$b_w + b_x$
3:	$[\![Y^1]\!] \leftarrow \mathsf{ActH}([\![U^1]\!], b_x)$	b_w
4:	**for** $i = 1$ **to** $n - 1$ **do**	-
5:	$[\![U^{i+1}]\!] \leftarrow [\![W^i]\!] \cdot [\![Y^i]\!]$	$2b_w$
6:	$[\![Y^{i+1}]\!] \leftarrow \mathsf{ActH}([\![U^{i+1}]\!], b_w)$	b_w
7:	**end for**	-
8:	$[\![U^{n+1}]\!] \leftarrow [\![W^n]\!] \cdot [\![Y^n]\!]$	$2b_w$
9:	$[\![Y^{n+1}]\!] \leftarrow \mathsf{ActO}([\![U^{n+1}]\!], 2b_w, b_y)$	b_y
10:	② 逆伝播の計算	-
11:	$[\![Z^{n+1}]\!] \leftarrow [\![Y^{n+1}]\!] - [\![T]\!]$	b_y
12:	$[\![Z^n]\!] \leftarrow \mathsf{ActH}'([\![U^n]\!], [\![Z^{n+1}]\!], [\![W^n]\!], b_y)$	b_w
13:	**for** $i = 1$ **to** $n - 1$ **do**	-
14:	$[\![Z^{n-i}]\!] \leftarrow \mathsf{ActH}'([\![U^{n-i}]\!], [\![Z^{n-i+1}]\!], [\![W^{n-i}]\!], b_w)$	b_w
15:	**end for**	-
16:	③ 勾配の計算	-
17:	$[\![G^0]\!] \leftarrow [\![Z^1]\!] \cdot [\![X]\!]$	$b_w + b_x$
18:	$[\![G^0]\!] \leftarrow \mathsf{rshift}([\![G^0]\!], b_x + H)$	b_w
19:	**for** $i = 1$ **to** $n - 1$ **do**	-
20:	$[\![G^i]\!] \leftarrow [\![Z^{i+1}]\!] \cdot [\![Y^i]\!]$	$2b_w$
21:	$[\![G^i]\!] \leftarrow \mathsf{rshift}([\![G^i]\!], b_w + H)$	b_w
22:	**end for**	-
23:	$[\![G^n]\!] \leftarrow [\![Z^{n+1}]\!] \cdot [\![Y^n]\!]$	$b_w + b_y$
24:	$[\![G^n]\!] \leftarrow \mathsf{rshift}([\![M^i]\!], b_y + H)$	b_w
25:	④ Adam によるパラメータ更新	-
26:	**for** $i = 0$ **to** n **do**	-
27:	$[\![M^i]\!] \leftarrow \beta_1 [\![M^i]\!] + (1 - \beta_1)[\![G^i]\!]$	$b_w + b_\beta$
28:	$[\![M^i]\!] \leftarrow \mathsf{rshift}([\![M^i]\!], b_\beta)$	b_w
29:	$[\![V^i]\!] \leftarrow \beta_2 [\![V^i]\!] + (1 - \beta_2)[\![G^i]\!] \circ [\![G^i]\!]$	$2b_w + b_\beta$
30:	$[\![V^i]\!] \leftarrow \mathsf{rshift}([\![V^i]\!], b_\beta)$	$2b_w$
31:	$[\![\hat{M}^i]\!] \leftarrow \hat{\beta}_{1,t} [\![M^i]\!]$	$b_w + b_{\hat{\beta}_1}$
32:	$[\![\hat{V}^i]\!] \leftarrow \hat{\beta}_{2,t} [\![V^i]\!]$	$2b_w + b_{\hat{\beta}_2}$
33:	$[\![\hat{G}^i]\!] \leftarrow \mathsf{Adam}([\![\hat{V}^i]\!], \eta', 2b_w + b_{\hat{\beta}_2}, b_{\hat{g}})$	$b_{\hat{g}}$
34:	$[\![\hat{G}^i]\!] \leftarrow [\![\hat{G}^i]\!] \circ [\![\hat{M}^i]\!]$	$b_{\hat{g}} + b_w + b_{\hat{\beta}_1}$
35:	$[\![\hat{G}^i]\!] \leftarrow \mathsf{rshift}([\![\hat{G}^i]\!], b_{\hat{g}} + b_{\hat{\beta}_1})$	b_w
36:	$[\![W^i]\!] \leftarrow [\![W^i]\!] - [\![\hat{G}^i]\!]$	b_w
37:	**end for**	-

出力層の活性化関数

表 4 中の ActO は，任意の出力層の活性化関数での処理に対応する．たとえば出力層の活性化関数が恒等関数の場合，表 4 手順 9 の $[\![Y^{n+1}]\!] \leftarrow \mathsf{ActO}([\![U^{n+1}]\!], 2b_w, b_y)$ は式 (23) のように identity を計算する．identity は式 (24) に示す処理を行う関数である．

$$[\![Y^{n+1}]\!] \leftarrow \mathsf{identity}([\![U^{n+1}]\!], 2b_w, b_y) \tag{23}$$

$$\mathsf{identity}([\![\vec{a}]\!], b_\alpha, b_\beta) := \mathsf{rshift}([\![\vec{a}]\!], b_\alpha - b_\beta) \tag{24}$$

出力層の活性化関数がソフトマックス関数だった場合，表 4 手順 9 の処理は $[\![Y^{n+1}]\!] \leftarrow \mathsf{softmax}([\![U^{n+1}]\!], 2b_w, b_y)$ となる．softmax は exp と reciprocal を用いて実現する．その際，式 (3) をそのまま計算するのではなく，式 (25) のように式 (3) の分母・分子をともに e^{u_i} で割ったものを計算する．具体的な処理を表 5 に示す．

$$\mathsf{softmax}(u_i) = \frac{1}{\sum_{j=0}^{k-1} e^{u_j - u_i}} \tag{25}$$

表 5 秘匿ソフトマックス関数計算アルゴリズム

入力 1：$[\![\vec{u}]\!] := [\![u_0]\!], [\![u_1]\!], \cdots, [\![u_{k-1}]\!]$
入力 2：$[\![u_i]\!]([\![\vec{u}]\!]$ の i 番目の要素)
出力：$\mathsf{softmax}([\![u_i]\!]) = [\![\frac{1}{\sum_{j=0}^{k-1} e^{u_j - u_i}}]\!]$

	計算式	精度 [bit]
1:	$[\![\vec{u} - u_i]\!] \leftarrow [\![\vec{u}]\!] - [\![u_i]\!]$	$2b_w$
2:	$[\![e^{\vec{u}-u_i}]\!] \leftarrow \mathsf{exp}([\![\vec{u} - u_i]\!], 2b_w, b_y)$	b_y
3:	$[\![\sum_{j=0}^{k-1} e^{u_j - u_i}]\!] \leftarrow \mathsf{sum}([\![e^{\vec{u}-u_i}]\!])$	b_y
4:	$[\![\frac{1}{\sum_{j=0}^{k-1} e^{u_j - u_i}}]\!] \leftarrow \mathsf{reciprocal}([\![\sum_{j=0}^{k-1} e^{u_j - u_i}]\!], b_y, b_y)$	b_y

3.3.2 Adam の処理

表 4 手順 33 の $[\![\hat{G}^i]\!] \leftarrow \mathsf{Adam}([\![\hat{V}^i]\!], \eta', 2b_w + b_{\hat{\beta}_2}, b_{\hat{g}})$ は sqrtinv を用いて実現する．Adam は式 (11) の $\frac{\eta}{\sqrt{\hat{v}}+\epsilon}$ を，$\hat{v}$ を秘匿したまま計算する関数である．本稿では $\epsilon = 0$ としており，sqrtinv に 0 が入力された場合は，sqrtinv の出力に関するメタ情報内で定めた最大値が出力される．

具体的に $[\![\hat{G}^i]\!] \leftarrow \mathsf{Adam}([\![\hat{V}^i]\!], \eta', 2b_w + b_{\hat{\beta}_2}, b_{\hat{g}})$ は以下のように処理する．なお $2b_w + b_{\hat{\beta}_2}$ は入力 $[\![\hat{V}^i]\!]$ の精度で，$b_{\hat{g}}$ は出力 $[\![\hat{G}^i]\!]$ の精度である．

$$[\![\hat{G}^i]\!] \leftarrow \mathsf{sqrtinv}([\![\hat{V}^i]\!], 2b_w + b_{\hat{\beta}_2}, b_{\hat{g}}) \tag{26}$$

$$[\![\hat{G}^i]\!] \leftarrow \mathsf{rshift}([\![\hat{G}^i]\!], \eta') \tag{27}$$

4. 実験

4.1 実験設定

MEVAL を用いて表 4 のアルゴリズムを実装し，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

全実験で共通の設定：ActH は ReLU とし，パラメータは He の初期化 [10] をしている．Adam のハイパーパラメータは $\beta_1 = 0.9$，$\beta_2 = 0.999$，$\eta = 2^{-10}(\eta' = 10)$，$\epsilon = 0$ とした．以下，クラス分類を行うもの (ActO が softmax) を提案手法 1，回帰分析を行うもの (ActO が identity) を提案手法 2 とする．明記していない場合は 1epoch ごとにシャッフルしており，処理時間はシャッフルやパラメータ初期化の時間も含む．実験結果は全て 5 回行った平均値である．

4.2 予測精度の評価方法

ソフトマックス関数を用いたクラス分類の予測精度の評価方法は，学習したパラメータを用いて得られた予測結果とラベルを比較して，予測結果の何%が正しかったかを予測精度とする一般的な方法である．

恒等関数を用いた回帰分析の予測精度の評価方法は，一般的な回帰分析の評価方法である決定係数 R^2 を用いる．学習したパラメータを用いて得られた予測結果を y，ラベルを t，ラベルの平均値を $\overline{y}$ とした時，決定係数 R^2 は下記のように計算できる．M は予測に用いたデータ，およびラベルの数である．

$$R^2 = 1 - \frac{\sum_{i=1}^{M}(t_i - y_i)^2}{\sum_{i=1}^{M}(t_i - \overline{y})^2} \tag{28}$$

4.3 提案手法 1：MNIST でのクラス分類

MNIST[5] は 0〜9 の手書き文字のデータセットで，784 個の画素値が入った学習用データと，それに対応する 0〜9 のラベルが 6 万件ずつある．またテスト用として，同じ形式のデータとラベルも 1 万件ずつ用意されている．予測精度はこのテスト用データを分類した結果とする．

モデル設定：$d_0 = 784$，$d_{n+1} = 10$

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]

4.3.1 先行研究・平文との性能比較

SecureML[14]，SecureNN[19]，ABY^3[13] および，筆者らが以前に実装したもの (以下 MHI19)[21] と提案手法 1 の性能を比較した結果を表 7 に示す．また平文との比較として，scikit-learn[6] の MLPClassifier を用いた場合の結果も記載した．$n = 2$，$d_1, d_2 = 128$，$m = 128$ 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	25,283	93.4
SecureNN	15	3,708	93.4
ABY^3	15	2,700	94
MHI19	1	412	94.2
提案手法 1	1	312	95.64
scikit-learn	1	17.7	95.54

reciprocal, exp, sqrtinv を用いたことによって，それらを全て秘密一括写像で処理していた MHI19 と比較して 100 秒 (約 1.3 倍) の高速化を達成した．また他者の先行研究で最も高速な ABY^3 より約 9 倍高速で，予測精度も上回った．

また，scikit-learn との処理時間の差は約 18 倍まで縮まった．両者ともデータ読み込みなどに要した時間も含んでおり，純粋に学習のみに要した時間は，提案手法 1 で 270 秒，scikit-learn では 1.8 秒であり，この部分のみに注目すると約 150 倍の差となっている

4.3.2 バッチサイズを変えた実験

$n = 2$，$d_1, d_2 = 128$ とし，バッチサイズ m を変えた場合の処理時間 (1epoch) を表 8 に示す．ミニバッチ処理時間

とは，1 つのミニバッチを処理するのに要した時間である．

表 8 各バッチサイズでの処理時間

バッチサイズ	64	128	256	512
処理時間 [s]	583	312	164	92
ミニバッチ処理時間 [s]	0.555	0.576	0.614	0.661

ミニバッチ処理時間は，バッチサイズによって大きく変化しないため，1epoch の処理時間はバッチサイズとほぼ反比例している．このことから，高速に処理したい場合はバッチサイズを大きくするのが有効だと考えられる．

4.3.3 ニューロン数を変えた実験

$n = 2$，$m = 128$ とし，隠れ層のニューロン数 d_1, d_2 を変えた場合の処理時間 (1epoch) を表 9 に示す．

表 9 各ニューロン数での処理時間

ニューロン数	32	64	128	256
処理時間 [s]	242	261	312	449

$d_1, d_2 = 256$ でも約 7.5 分という実用的な処理時間になった．この結果の傾向から，より大規模なネットワークでも，提案手法は実用的な時間で処理できると考えられる．

4.3.4 隠れ層の数を変えた実験

$m = 128$，$d_1, d_2 = 128$ とし，隠れ層の数 n を変えた場合の処理時間 (1epoch) を測定した結果を表 10 に示す．

表 10 各隠れ層の数での処理時間

隠れ層の数	1	2	3	4
処理時間 [s]	278	312	336	367

隠れ層を 4 層にした場合でも 6 分程度であり，十分に実用的な処理時間である．また 1 層増える毎に約 30 秒ずつ増加している傾向から，より深いネットワークにした場合の処理時間も大まかに見積もることができ，更に層を増やしても実用的な時間内で処理できることが予想できる．

4.3.5 平文との予測精度の比較

固定小数点数での処理による数値誤差が，どの程度予測性能に影響するか調べるため，1epoch ごとのシャッフルをせず，パラメータ初期値を固定して，浮動小数点数で同様の処理をした場合 (平文) の予測精度と比較した．シャッフルせず，パラメータ初期値を固定した場合，数値誤差が無ければ同じ結果が得られる．$m = 16$，$d = 20$，$n = 2$ の設定で行った結果を表 11 に示す．

表 11 予測精度の比較

	平文	提案手法 1
予測精度 [%]	94.65	94.44

平文と提案手法 1 の予測精度の差は 0.2%となった．ディープラーニングでは，パラメータの初期化やミニバッチ確率的勾配降下法によって，同じ形のネットワークや同じデータセットで学習しても結果はランダムであり，それによって予測精度が 1%以上変わることも多い．そのため，数値誤差による差よりも学習結果のランダムさによる差のほうが大きく，固定小数点数での処理によって学習結果が劣化する可能性は低いと考えられる．

4.4 提案手法 2：Boston データでの回帰分析

Boston データはボストン市の住宅価格を目的変数とし，犯罪発生数や NOx の濃度など 13 の説明変数がある，データ数 506 件のデータセットである [1]．データセットが小さいため，学習と予測それぞれに全データを用いて評価した．

モデル設定：$d_0 = 13$，$d_{n+1} = 1$

精度設定：$b_w = 20$[bit]，$b_x = 10$[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]

4.4.1 隠れ層の数を変えた実験

$m = 16$，$d = 20$ とし，隠れ層の数 n を変化させた場合の処理時間・決定係数 R^2 を表 12 に示す．また平文との比較として，scikit-learn の MLPRegressor を用いた場合の結果も記載した．参考までに，MLPRegressor で $n = 2$ とした場合の処理時間は 0.605 秒であった．

表 12 各隠れ層の数での処理時間・決定係数 R^2

隠れ層の数	2	3	4
処理時間 [s](提案手法 2)	494	566	640
R^2(提案手法 2)	0.8033	0.8593	0.9028
R^2(scikit-learn)	0.8076	0.8614	0.8839

$n = 4$ の場合でも処理時間は 11 分弱，かつ $R^2 = 0.9$ と平文と遜色なく高い予測性能が得られた．これにより，提案手法の ActO を変えるだけで，クラス分類だけでなく回帰分析でも，高速に良い学習結果が得られることが分かった．

4.4.2 平文との決定係数の比較

提案手法 1 での実験と同様に，固定小数点数での処理による決定係数 R^2 への影響を調べた．$m = 16$，$d_1, d_2 = 20$，$n = 2$ の設定で行った結果を表 13 に示す．

表 13 決定係数 R^2 の比較

	平文	提案手法 2
R^2	0.7972	0.8016

平文と提案手法 2 の R^2 の差は 0.0044 となり，提案手法 1 での実験と同様，パラメータ初期化などのランダムさによる影響の方が大きいため，固定小数点数での処理による数値誤差で学習結果が劣化する可能性は低いと考えられる．

4.4.3 提案手法 1 と提案手法 2 の比較

提案手法 1 と提案手法 2 で ActO 以外を全て揃えて，1epoch の処理時間を比較した結果を表 14 に示す．

モデル設定：$m = 128$, $n = 2$，$d_0 = 784, d_1, d_2 = 128$
$d_{n+1} = 10$(提案手法 1)，$d_{n+1} = 1$(提案手法 2)

表 14 提案手法 1 と提案手法 2 の処理時間 [s]

	提案手法 1	提案手法 2
処理時間 [s]	312	212

ActO 以外は同じ処理のため，表 14 での差は softmax の計算によるもので，想定通りの結果となった．

4.5 100 属性 ×1000 万件データでの実験

100 属性 ×1000 万件の大規模データを用いて，提案手法 1・2 の処理時間 (1epoch) を測定した結果を表 15 に示す．なお，精度設定は提案手法 1 の実験と同じである．

モデル設定：$n = 2$，$d_0 = 100$，$d_1, d_2 = 128$，$d_{n+1} = 10$(提案手法 1)，$d_{n+1} = 1$(提案手法 2)

表 15 100 属性 1000 万件データの処理時間 [s]

バッチサイズ	提案手法 1	提案手法 2
512	11,666(約 3.2[h])	8,391(約 2.3[h])
1024	7,135(約 2.0[h])	5,783(約 1.6[h])

100 属性 ×1000 万件の大規模データでも，提案手法 1・2 ともに 2〜3 時間程度で処理できており，数 epoch 学習する場合でも夜間バッチ等で対応出来る範囲であるため，実用的な処理性能であると言える．

5. おわりに

本稿の成果は以下の 3 つである．

(1) 秘密計算ディープラーニングの更なる高速化

(2) より大きなネットワークへの対応，分析の拡充

(3) 大規模な学習データでの実用性を確認

筆者らが以前に実装した秘密計算ディープラーニング [21] は他者の先行研究より 6 倍以上高速だったが，本稿では指数や除算など高コストな処理を新たな関数にしたことで，更なる高速化を達成した．また，先行研究では決まった形のニューラルネットワークでのみ実験されてきたが，本稿ではニューロン数・隠れ層の数などを変えて実験し，大きなネットワーク・深いネットワークでの実用性も示した．先行研究では固定となっていた出力層の活性化関数も自由に設定できるアルゴリズムを提案し，クラス分類・回帰分析の両方で，高い処理性能・予測性能を実現した．そして，100 属性 ×1000 万件の大規模データでも実用的な時間で学習できることを示した．以上の成果より，本稿の秘密計算ディープラーニングは高い実用性を有すると言える．

5.1 今後の展望

秘密計算ディープラーニングを実運用する具体的な適用先や運用方法について検討し，そこから新たな研究課題を抽出することで更なる改良を目指す．また，それと並行して秘密計算 AI ライブラリの実現に向け，これまでに取り組んだロジスティック回帰やディープラーニングの改良だけでなく，他の AI 手法の検討・実装にも取り組んでいく．

参考文献

[1] Boston housing. https://archive.ics.uci.edu/ml/machine-learning-databases/housing/.

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

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

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

[5] MNIST database. http://yann.lecun.com/exdb/mnist/.

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

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

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] Payman Mohassel and Peter Rindal. ABY 3: a mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 35–52. ACM, 2018.

[14] 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.

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

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

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

[18] 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.

[19] 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.

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

[21] 三品気吹, 濱田浩気, 五十嵐大. 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. In *FIT*, 2019.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.