Computer Security Symposium 2018
22 - 25 October 2018

ラウンド効率のよい2者間秘匿計算とその秘匿畳み込みニューラルネットワークへの応用

大畑 幸矢[1]

概要：秘密分散に基づく秘匿計算は高いスループットを達成できることから注目されているが、非線形関数のような「深い」回路を計算する際にはラウンド数が多くなってしまい、特に通信遅延が大きい環境の下では性能が大幅に低下してしまうという問題がある。本稿では秘密分散に基づく 2 者間秘匿計算において、事前計算の工夫により多入力の論理積ゲートを 1 ラウンドで計算する手法と、それに基づくラウンド効率のよいプロトコル（一致判定、大小比較、最大値抽出など）の構成法を提案する。また、そのような秘匿計算のシミュレータを実装して (1) ゲート自体や構成したプロトコルの性能評価、及び (2) 深層畳み込みニューラルネットワークの秘匿推論及び秘匿訓練に用いられる関数の性能評価を行う。

キーワード：二者間秘匿計算、多入力論理積ゲート、深層畳み込みニューラルネットワーク

Round-Efficient Secure Two-Party Computation and Its Application to Privacy-Preserving Convolutional Neural Networks

Satsuya Ohata[1]

Abstract: Secure multi-party computation has received much attention in recent years. In this paper, we propose round-efficient secure 2-party protocols via a multi-fan-in AND gate. We implement a simulator for those protocols and functions for convolutional neural networks, and evaluate their efficiency.

Keywords: Secure Two-Party Computation, Multi-Fan-in AND Gate, Deep Convolutional Neural Networks

1. はじめに

1.1 秘匿計算

秘匿計算（Multi-Party Computation, MPC）[15], [32] は通信やデータ保存における情報漏洩を防止する従来の暗号とは異なり、データ処理や管理の過程での情報漏洩をも防止することに貢献しうる技術である。秘匿計算においては（完全）準同型に基づく手法、ガーブルド回路に基づく手法、秘密分散に基づく手法などがあり、それぞれセッティングが異なったり計算する回路に得手不得手があったりするものの、プロトコルの工夫や実装の高度な最適化によって高いスループットを出せることがわかってきた秘密分散ベース MPC が注目を集めている。本稿でもこの秘密分散ベース MPC を扱う。

近年登場した高速な秘密分散ベース MPC は 2-out-of-3 秘密分散に基づく 3 者秘匿計算（3PC）が多い。これは効率の良い 2 者間秘匿計算（2PC）においては Beaver Triple の計算（事前計算と呼ぶ）に準同型暗号や紛失通信が必要であり、その計算や通信に要する時間が膨大である一方で、3PC はそれに相当する計算をローカルの軽量な計算に置き換えられることが理由の一つであると思われる。ただし、高速な MPC を提案している論文 [1] の元論文 [12] でも述べられているように、実用上より大切なのは 2PC である[*1]。

[1] 国立研究開発法人 産業技術総合研究所, 135-0064 東京都江東区青海 2-4-7, National Institute of Advanced Industrial Science and Technology (AIST) (satsuya.ohata@aist.go.jp)

[*1] 2PC が 3PC と全く同じ効率・速度で動作するのであれば、サーバ・通信回線を準備するコストや消費電力の面などで 2PC にメリットがあると思われる。

2PC における重い事前計算を回避する工夫として、事前計算のみを行う第三者の存在を仮定する Server-aided モデル [21] や、委託計算の文脈でデータ処理を委託するクライアントが事前計算を行う Client-aided モデル [23], [24] が提案されている。これらの方式では事前計算において準同型暗号/紛失通信が不要であり、トータルの処理時間を大幅に短縮することに成功している[*2]。本稿でもこれらのモデルを採用することを前提に議論を進める。

1.2 既存研究における課題

秘密分散ベース MPC は高い性能を出せることが実証されているが、その性能はプロトコルや実装だけでなく、良好な通信環境に拠るところも大きい。秘密分散ベース MPC は非線形演算などの深い回路を計算すると通信ラウンド数が多くなってしまうという問題があるが、ほとんどの既存研究が前提にしている 1Gbps（あるいは 10Gbps）の LAN 環境がこの問題による悪影響を目立たなくしている。LAN 環境を想定して問題ない状況も存在するが、MPC の適用範囲拡大を考慮すると、通信遅延の大きい WAN 環境で良い性能を出せる MPC が望ましい[*3]。

また、すべての処理をベクトル化して一括実行できるようなケースでは通信遅延の影響が相対的に小さくなるが、現実のアプリケーションはそのような状況ばかりとは限らない。例えば深層学習におけるミニバッチ学習のように、ミニバッチサイズとモデルの性能間にトレードオフがある場合や、データが逐次的にやってくる場合などが考えられる。実際、後述するシミュレータで適当なパラメータを設定して実験してみても、WAN 環境ではかなり多量の入力を同時処理しない限りは遅延に起因する通信時間が大部分を占める。すなわち、ゲートレベルを超えた具体的なプロトコルの構成においては、いかに回路を浅くできるか（＝ラウンド数を減らせるか）が重要な研究課題となる。

1.3 本論文の貢献

本論文の貢献は以下の 2 点である。

(1) 通信ラウンド数の少ない秘密分散ベースの 2 者間秘匿計算プロトコルを提案する。核になるのは 2 入力の算術積/論理積ゲートを計算するために必要な Beaver Triple を $N(>2)$ 入力ゲート向けに拡張した「拡張 Beaver Triple」[*4]である。拡張 Beaver triple を用いると任意の N に対して N-fan-in の AND ゲートを 1 ラウンドで計算可能であるが、生成及びゲート計算の時間計算量・空間計算量はともに N の指数アルゴリズムとなるため、N の大きさは制限する必要がある。本稿では $N \leq 8$ の N-fan-in-AND ゲートを用いることとし、それらを用いた 2 者間秘匿計算プロトコル（一致判定、大小比較、最大値抽出など）を提案する。提案するプロトコルは必要な通信ラウンド数が少ない。例えば大小比較は（64bit 空間においては）4 ラウンドで実行でき、これは最もラウンド効率のよい既存研究結果 [24] の 5 ラウンドよりもさらに少なくなっている。また、提案するゲート/プロトコルのシミュレータを実装して性能評価を行った結果、メモリを多く消費するものの（少なくとも WAN 環境においては）ラウンド数削減の効果が計算時間の増加分を上回ることを確認した。

(2) 上記のプロトコルを利用して畳込みニューラルネットワークの計算に必要な関数を構築し、適当なパラメータを振ってモデルの推論および訓練に要する時間をシミュレーションした。畳み込みニューラルネットワークは秘匿計算の対象としてはかなり厳しいアプリケーションであるが、シミュレーションの結果、推論は WAN 環境でもオンライン計算は許容できる時間内で完了することがわかった。また、訓練は小数の扱いや高コストな除算が原因でまだ現実的とは言えないこともわかった。

1.4 本稿の構成

2 章では表記法・前提知識・関連研究を示す。3 章では拡張 Beaver triple とそれを用いたプロトコルの構成を提案する。4 章では提案する秘匿計算の性能評価を行う。5 章では畳み込みニューラルネットワークの秘匿計算を議論する。6 章は本稿のまとめである。

2. 準備及び関連研究

2.1 表記

$\mathbb{Z}_{2^n}$ 上の 2-out-of-2 秘密分散（以降 $(2,2)$-SS と表記）は 2 つのアルゴリズム Share と $\mathsf{Reconst}$ からなる。分散アルゴリズム Share は $x \in \mathbb{Z}_{2^n}$ を入力に取り、2 つのシェア $(\llbracket x \rrbracket_1, \llbracket x \rrbracket_2) \in \mathbb{Z}_{2^n}^2$ を出力する。簡単のため、$\llbracket x \rrbracket$ を $(\llbracket x \rrbracket_1, \llbracket x \rrbracket_2)$ の意味で用いる。再構成アルゴリズム $\mathsf{Reconst}$ は $\llbracket x \rrbracket$ を入力に取り、x を出力する。

2.2 GMW プロトコル

GMW プロトコル [15] は 2 者間プロトコルであり、以下のようなシェアを用いる。

- $\mathsf{Share}(x)$: ランダムな $r \in \mathbb{Z}_{2^n}$ を選び、$\llbracket x \rrbracket_0^{\mathsf{A}} = r$ と $\llbracket x \rrbracket_1^{\mathsf{A}} = x - r$ を出力する。

[*2] ただし、計算結果の正当性を保証するためには事前計算を行うサーバやクライアントが不正をしないことを仮定する必要がある。また、オンライン計算は 2PC であるものの別の存在が登場するため、これを 2PC と呼ぶのが適切かは微妙である。

[*3] LAN 環境であれば例えば同じ建物内にあるサーバ間通信や同じ業者の（クラウド）サーバ間通信などといった状況が想定されるが、この状況で「各サーバは結託しないのが前提である」と主張するのはやや苦しいのではないかと著者は考えている。

[*4] 3 つ組ではないのでやや不適切な呼び名だが、拡張紛失通信のイメージから本稿ではこのように呼ぶことにする。

- $\mathsf{Reconst}(\llbracket x \rrbracket^{\mathsf{A}}_0, \llbracket x \rrbracket^{\mathsf{A}}_1)$: $\llbracket x \rrbracket^{\mathsf{A}}_0 + \llbracket x \rrbracket^{\mathsf{A}}_1$ を出力する。

右上の A は Arithmetic（算術）シェアであることを意味する。これらのシェアの上での加算 $\mathbf{ADD}(x, y) := x + y$ は、単にシェア同士を足すだけで実現できる。乗算 $\mathbf{MULT}(x, y) := xy$ は様々な実現方法があるが、本稿では Beaver triple に基づく手法 [3] を採用する。Beaver triple とは $a_0 + a_1 = a$、$b_0 + b_1 = b$、$c_0 + c_1 = c$、$ab = c$ という 4 つの条件を満たす 3 つ組乱数 (a_0, b_0, c_0) および (a_1, b_1, c_1) のことで、乗算プロトコルへの入力と関係なく事前に計算しておける。パーティ P_0 は $\llbracket x \rrbracket^{\mathsf{A}}_0 - a_0$ と $\llbracket y \rrbracket^{\mathsf{A}}_0 - b_0$ を、パーティ P_1 は $\llbracket x \rrbracket^{\mathsf{A}}_1 - a_1$ と $\llbracket y \rrbracket^{\mathsf{A}}_1 - b_1$ をそれぞれ計算して送り合った後、互いのローカルで $\mathsf{Reconst}(\llbracket x \rrbracket^{\mathsf{A}}_0 - a_0, \llbracket x \rrbracket^{\mathsf{A}}_1 - a_1)$ 及び $\mathsf{Reconst}(\llbracket y \rrbracket^{\mathsf{A}}_0 - b_0, \llbracket y \rrbracket^{\mathsf{A}}_1 - b_1)$ を実行して $x' = x - a$ 及び $y' = y - b$ を再構成する。その後、P_0 は $\llbracket z \rrbracket^{\mathsf{A}}_0 = x'y' + x'b_0 + y'a_0 + c_0$ を、P_1 は $\llbracket z \rrbracket^{\mathsf{A}}_1 = x'b_1 + y'a_1 + c_1$ を計算すると、$\mathsf{Reconst}(\llbracket z \rrbracket^{\mathsf{A}}_0, \llbracket z \rrbracket^{\mathsf{A}}_1) = xy$ となる。乗じる値が公開値 c の場合は、それぞれがローカルで自身のシェアを c 倍すればよい。GMW プロトコルは semi-honest 安全であることが証明されており、これらの組み合わせで構成するプロトコルの semi-honest 安全性は Composition Theorem [14] より成立するため、本稿ではこれ以降安全性に関して議論しない。また、加算を排他的論理和に置き換えることで、論理ゲート（**XOR** および **AND**）も計算可能である。本稿では論理（Boolean）シェアを $\llbracket x \rrbracket^{\mathsf{B}}$ と書くことにする。また、論理否定 **NOT** は $\mathbf{NOT}(\llbracket x \rrbracket^{\mathsf{B}}) = (\tilde{}\llbracket x \rrbracket^{\mathsf{B}}_0, \llbracket x \rrbracket^{\mathsf{B}}_1)$ で、論理和 **OR** は $\mathbf{OR}(\llbracket x \rrbracket^{\mathsf{B}}, \llbracket y \rrbracket^{\mathsf{B}}) = \tilde{}\mathbf{AND}(\tilde{}\llbracket x \rrbracket^{\mathsf{B}}, \tilde{}\llbracket y \rrbracket^{\mathsf{B}})$ で実現できる。

2.3 畳み込みニューラルネットワーク

パーセプトロンを重ねて層を形成したものがニューラルネットワークであり、ある入力に対して各層間の伝搬を順に計算し、最終層の結果を出力として返す。入力 x、重み w、バイアス b に対し、出力 y は $y = w \cdot x + b$ と計算され、教師データを用いて w, b の値を適切な値に調整することを訓練と呼ぶ。この出力に活性化関数と呼ばれる非線形関数を適用して次の層の入力とする。活性化関数としてはランプ関数 $ReLU(x) = \max(0, x)$ [25] が用いられることが多い。これらの層を多く積むことでネットワークの非線形性を高め、大きな表現力を得ている。ニューラルネットワークで分類問題を解く際には、Softmax 関数と呼ばれる、入力を確率として解釈できるようにする関数の層を入れる。10 クラス分類の場合、出力 $\mathbf{y} = \{y_0, \cdots, y_{10}\}$ を $\sum_{i=0}^{10} y_i = 1$ かつ $0 \leq y_i \leq 1$ とし、$argmax(\mathbf{y})$ が入力が属するクラスとなるように訓練を行う。

畳み込みニューラルネットワーク（Convolutional Neural Networks, CNN）[11], [18] はニューラルネットワークに畳み込み層を加えたものである。畳み込み層ではフィルタによって入力の各領域を一つのスカラー値に畳み込む操作を行った後、いくつかの領域の値を一つ（多くの場合は最大値）に集約するプーリングと呼ばれる操作を入れることが多い。CNN においては特徴量の自動的な獲得を期待することができ、多くのタスクで良い性能を出すことがわかってきたため、CNN の新しいアーキテクチャが多く提案されている。

2.4 関連研究

2.4.1 秘密分散に基づく MPC

秘密分散に基づく MPC は Sharemind 社の研究 [4], [30] が古くから有名であったが、Araki ら [1], [2] や Chida ら [8] などによる $(2, 3)$-SS に基づく高速な MPC が近年提案されている。ゲートレベルの高速化だけでなく、算術シェアと論理シェア（や Yao シェア）の効率的な変換プロトコル [10], [22], [28] や、大小比較などのより高機能なプロトコル [24]、具体的な秘匿計算アプリケーションの設計と実装（最近は機械学習が多いので次の節で紹介する）といった研究が進められている。3 者計算 [1] においては事前計算（Beaver triple ではなく correlated randomness と呼ばれる）が各サーバのローカル計算で済む（＝準同型暗号や紛失通信が不要）なため、2 者計算と比べて非常に高速である。2 者計算においては、オンライン計算を担うサーバ以外の存在が Beaver Triple を生成・配布してよいことにすると、（準同型暗号/紛失通信が不要なため）非常に事前計算が高速化される。事前計算を担う専用サーバを仮定するモデルは Server-aided モデル [21]、委託計算の文脈で委託するクライアントがデータの分割だけでなく事前計算も行うモデルを Client-aided モデル [23], [24] と呼ぶ。

2.4.2 秘匿機械学習

準同型暗号/ガーブルド回路/秘密分散を用いて、データを隠したまま機械学習を行う研究が活発になされている。2015 年以前は様々な学習器が対象になっていた [5], [16] が、2016 年以降はニューラルネットの秘匿計算をメインに据えた研究 [6], [7], [13], [17], [19], [22], [23], [28], [29], [31] が多い。訓練は平文で実施済みという前提で推論処理を秘匿計算するものが多く、訓練の秘匿計算を扱った研究例 [17], [22], [23], [31] は少ない。理由はまだ高速処理が難しいからであり、例えば準同型暗号ベースの [17] では計算途中にノイズが溜まってしまった暗号文をクライアントに戻して復号および再暗号化させる必要がある。他の結果においてもあまり実用的とは言えない時間がかかっている訓練の秘匿計算はまだ難しいのが現状である[*5]。

[*5] [22] のみ、CNN の秘匿訓練が 1 時間で終了したと主張しているが、（少なくとも eprint 版では）畳み込み演算を全結合層の計算で overestimate したと書かれており、これが何を意図しているかは不明である。また、プーリング層を省略したり、秘匿計算向けの小さなパラメータ（フィルタ枚数は 5 枚、畳み込みは 1 層のみ、等）を設定していることが計算時間の短縮に効いている。

3. ラウンド効率のよい二者間秘匿プロトコル

3.1 拡張 Beaver Triple と多入力論理積ゲート

2.2 節の Beaver triple を用いた **MULT** ゲート計算を再度見てみることにする。$[\![z]\!]_0^{\mathsf{A}} = x'y' + x'b_0 + y'a_0 + c_0$、$[\![z]\!]_1^{\mathsf{A}} = x'b_1 + y'a_1 + c_1$ であるから、

$$\begin{aligned}&[\![z]\!]_0^{\mathsf{A}} + [\![z]\!]_1^{\mathsf{A}}\\ &= (x-a)(y-b) + x'(b_0+b_1) + y'(a_0+a_1) + (c_0+c_1)\\ &= (xy - bx - ax + ab) + (x-a)b + (y-b)a + c\\ &= xy - ab + c\end{aligned}$$

となり、$c = ab$ であるから xy が残る。すなわち、$(x-a)(y-b)$ から出てくる xy 以外の項を Beaver triple によってうまく消去できるような仕組みになっているということである。ここで説明したのは 2 入力のゲートであるが、これを 3 入力に拡張することを考える。すなわち、$(x-a)(y-b)(z-c)$ から出てくる xyz 以外の項をうまく消去できるような Beaver triple の作り方を考える。これは比較的自然な拡張で構成することができて、3 入力 **MULT** ゲートを計算するためには、$a_0+a_1=a$、$b_0+b_1=b$、$\cdots$、$g_0+g_1=g$ かつ

- $d=ab$、$e=bc$、$f=ca$
- $g=abc$

を満たす 7 つ組を用いて

(1) P_i は $[\![x]\!]_i^{\mathsf{A}} - a_i$、$[\![y]\!]_i^{\mathsf{A}} - b_i$、$[\![z]\!]_i^{\mathsf{A}} - c_i$ を計算して相手に送り、

(2) 互いに $x' = x-a$、$y' = y-b$、$z' = z-c$ を再構成し、

(3) P_0 は $[\![w]\!]_0^{\mathsf{A}} = x'y'z' + x'y'c_0 + y'z'a_0 + z'x'b_0 + x'e_0 + y'f_0 + z'd_0 + g_0$ を、P_1 は $[\![w]\!]_1^{\mathsf{A}} = x'y'c_1 + y'z'a_1 + z'x'b_1 + x'e_1 + y'f_1 + z'd_1 + g_1$ を計算する。

この時、$\mathsf{Reconst}([\![w]\!]_0^{\mathsf{A}}, [\![w]\!]_1^{\mathsf{A}}) = xyz$ となっている。4 入力以上でも同じように拡張することで多入力ゲートを構成でき、算術積 **MULT** でなく論理積 **AND** でも $+$ と $-$ を $\oplus$ に置き換えるだけで全く同様のことが言える。本稿ではこれ以降、多入力ゲート向けの Beaver triple を（もはや 3 つ組ではないが）「拡張 Beaver triple」と呼ぶことにする。Damgård らの手法 [9] でも多入力ゲートを取り扱っているが、拡張 Beaver triple を用いた手法は彼らが 2 ラウンドを要するのに対して 1 ラウンドで済み、また平文の空間が逆元を持たなくてもよいためプロトコル設計上使い勝手のよい平文空間 $\mathbb{Z}_{2^n}$ を使える。ただし欠点もある。

拡張 Beaver triple の欠点：3 入力用の拡張 Beaver triple を見てわかるように、2 入力の場合は 3 つ組だったものが 3 入力で 7 つ組に増加している。N 入力の場合を考えると必要になるのは $\sum_{j=1}^{N} {}_N C_j = (2^N - 1)$ つ組ということになり、メモリ消費量、拡張 Beaver triple 生成の計算量、N 入力ゲートの計算量が指数的に増加する。よって、N の大きさは制限する必要があり、本稿では $N \leq 8$ のゲートのみを扱ってプロトコルの構成を考えることにする[*6]。

3.2 提案する秘匿計算プロトコル

算術積 **MULT** は多入力ゲートを計算できてもそれほど大きな利点はないが、論理積 **AND** の多入力ゲートは、高機能なプロトコルを少ないラウンド数で構成する際に非常に有用なツールである。紙面の都合でいくつかのプロトコルに関しては構成の概略のみになってしまうが、以下で具体的なプロトコルの構成について述べる。基本的には Sharemind による構成 [4], [30] を多入力ゲートによって効率化したものが多い。以下では平文空間は $\mathbb{Z}_{2^n}$ とし、もともと 1 ラウンドの内積 **DotProduct** は説明を省略する。

3.2.1 一致判定

一致判定プロトコル $\mathbf{Equality}([\![x]\!]^{\mathsf{A}}, [\![y]\!]^{\mathsf{A}})$ は $x=y$ であれば $z=1$、そうでなければ $z=0$ の論理シェア $[\![z]\!]^{\mathsf{B}}$ を出力するプロトコルである。[30] の構成では P_0 が $[\![x]\!]_0^{\mathsf{A}} - [\![y]\!]_0^{\mathsf{A}}$ を、P_1 が $[\![y]\!]_1^{\mathsf{A}} - [\![x]\!]_1^{\mathsf{A}}$ それぞれローカルで計算し、その算術値を 2 進展開した後、全てのビットの **OR** を計算し、その否定を取っている。$x=y$ かつその時に限り $[\![x]\!]_0^{\mathsf{A}} - [\![y]\!]_0^{\mathsf{A}}$ と $[\![y]\!]_1^{\mathsf{A}} - [\![x]\!]_1^{\mathsf{A}}$ は等しくなり、全てのビットの **OR** が 0 を出力するので最後にそれを反転すればよい、という仕組みである。通信が発生するのは **OR** の計算であり、2 入力の **AND** ゲートを用いるとツリー状に計算しても $\mathcal{O}(\log n)$ ラウンドの通信、例えば $n=16$ であれば 4 ラウンドを要する。

N 入力 **AND** ゲート（から構成した N 入力 **OR** ゲート）を用いてよいのであれば、この 4 ラウンドを 2 ラウンドに減らすことができる。16 ビットの入力を 4 ビット $\times 4$ 個に分割し、それぞれに 4 入力 **OR** を適用して 4 ビットの出力を得た後、それをさらに 4 入力 **OR** で処理すればよい。最初の 4 回の 4 入力 **OR** ゲートは並列に実行できるため、トータル 2 ラウンドで計算できる。より正確には、入力数 8 までの **OR** ゲートを使ってよいのであれば、必要な通信ラウンド数は $\lceil \frac{\log n}{3} \rceil$ であり、すなわち $n \leq 8$ のとき 1 ラウンド、$8 < n \leq 64$ のとき 2 ラウンドで **Equality** を計算可能である。

3.2.2 非零最上位ビット抽出

非零最上位ビット抽出プロトコル $\mathbf{MSNZB}([\![x]\!]^{\mathsf{B}})$ は、x のビットのうち、最上位の非零ビットの箇所にのみ 1 が立っている z の論理シェア $[\![z]\!]^{\mathsf{B}}$ を出力するプロトコルであり、この後構成する桁あふれ判定や最大値抽出などの重要な構成要素となる。[4] や [30] で提案されているのは Prefix OR を用いた手法である。Prefix OR とは、まず x

[*6] そこまでするなら通信ラウンド数の少ないガーブルド回路に変換して計算すればいいのではと考える読者もいると思われるが、ガーブルド回路に変換するコストなどを考慮するとたとえ非線形関数でも論理回路の構成を工夫して計算するほうが高速であると主張している論文 [31] もあり、本稿もその立場を取る。

Algorithm 1 $n = 16$ で 2 ラウンドの MSNZB プロトコル

Functionality: $[\![z]\!]^{\mathsf{B}} \leftarrow \mathbf{MSNZB}([\![x]\!]^{\mathsf{B}})$

Input: 16 ビットの論理シェア $[\![x]\!]^{\mathsf{B}}$、$[\![x]\!]^{\mathsf{B}}$ の各ビットを $[x_{15}, x_{14}, \cdots, x_0]$ と書くことにする

Output: 16 ビットの論理シェア $[\![z]\!]^{\mathsf{B}}$、z は x の非零最上位ビットのみ 1 でそれ以外は 0 となる値

1: $x'_{15} = x_{15}$
2: $x'_{14} = \mathbf{OR}2(x_{14}, x_{15})$
3: $x'_{13} = \mathbf{OR}3(x_{13}, x_{14}, x_{15})$
4: 以降 $i = 12, ..., 0$ に対して $x'_i = \mathbf{OR}4(x_i, x_{i+1}, x_{i+2}, x_{i+3})$
5: $i = 0, 1, 2, 3$ に対して $z_{15-i} = x'_{15-i}$
6: $i = 0, 1, 2, 3$ に対して $z_{11-i} = \mathbf{OR}2(x'_{11-i}, x'_{15-i})$
7: $i = 0, 1, 2, 3$ に対して $z_{7-i} = \mathbf{OR}3(x'_{7-i}, x'_{11-i}, x'_{15-i})$
8: $i = 0, 1, 2, 3$ に対して $z_{3-i} = \mathbf{OR}4(x'_{3-i}, x'_{7-i}, x'_{11-i}, x'_{15-i})$
9: $[\![z]\!]^{\mathsf{B}} = [z_{15}, \cdots, z_0]$
10: $[\![z]\!]^{\mathsf{B}} = [\![z]\!]^{\mathsf{B}} \oplus ([\![z]\!]^{\mathsf{B}} \gg 1)$
11: **return** $[\![z]\!]^{\mathsf{B}}$

を 1 ビット右シフトしたものと x 自身との **OR** を計算し、その結果を x' とする。次に x' を 2 ビット右シフトしたものと x' 自身との **OR** を計算し... という計算を繰り返す。i 回目には 2^{i-1} ビット右シフトとの **OR** 演算が走り、この操作を $\log n$ 回繰り返すと最上位非零ビットより右側のビットが全て 1 で埋まることが保証される。この値を X とすれば、X を 1 ビット右シフトしたものと X 自身の **XOR** を計算することで、非零最上位ビットの箇所のみ 1、それ以外は全て 0 の出力が得られる。

通信が発生するのは $\log n$ 回の **OR** を計算する Prefix OR であるが、これも N 入力 **OR** ゲートがあればラウンド数を減らすことが可能である。$n = 16$ のケースで 4 入力の **OR** ゲートを用いた 2 ラウンドの **MSNZB** プロトコルを Algorithm 1 に示す。**OR**N は N 入力 **OR** ゲートの意味である。Algorithm1 において、1〜4 と 5〜8 は並列に実行できるためそれぞれ 1 ラウンドであり、9,10 の操作では通信が発生しないため、必要な通信ラウンド数は 2 である。通常の Prefix OR においては i ラウンドで最上位非零ビットの右側 $2^i - 1$ ビットまで 1 で埋まることが保証されているが、**OR**4 を使うことで i ラウンドで最上位非零ビットの右側 $4^i - 1$ ビットまで 1 で埋まることが保証される。**OR**8 を使えば $n = 8$ までの場合は 1 ラウンド、$n = 64$ までの場合は 2 ラウンドで **MSNZB** を計算できる。

3.2.3 桁あふれ判定、ビット抽出、大小比較

桁あふれ判定プロトコル **Overflow**、ビット抽出プロトコル **ExtractBit**、大小比較プロトコル **Comparison** はいずれも [4], [30] と同様なので省略する。**MSNZB** のラウンド数が改善された効果でこれらのプロトコルもラウンド数が改善され、**Overflow** と **ExtractBit** は 3 ラウンド、**Comparison** は 4 ラウンドで計算が可能である。特に **Comparison** プロトコルは既存研究の中で最もラウンド数が少ないもの [24] でも 5 ラウンドを要しており、提案するプロトコルはそれよりもさらに 1 ラウンド少ない。

3.2.4 論理算術シェア変換

論理算術シェア変換 **B2A** は、互いが持っている論理シェアを秘密だと思って算術シェアに分割して片方を相手に送った後、$[\![x]\!]^{\mathsf{A}}_0 + [\![x]\!]^{\mathsf{A}}_1 - 2 \cdot \mathbf{MULT}([\![x]\!]^{\mathsf{A}}_0, [\![x]\!]^{\mathsf{A}}_1)$ を計算すればよい。必要な通信ラウンド数は 2 であり、多入力 **AND** ゲートがあっても無くてもラウンド数は変わらない。

3.2.5 右シフト

算術シェア入力の右シフトプロトコル **RightShift** は [4] と同様なので省略する。**Overflow** が 2 並列で 1 回、**B2A** が 2 並列で 1 回走るので、合計 $3 + 2 = 5$ ラウンドで計算できる。

3.2.6 最大値（最小値）抽出

最大値抽出プロトコル **Max**(**x**) は、ベクトル **X** の要素の中から最大のものを出力するプロトコルである。例えば $\mathbf{x} = [4, 10, 2, 7]$ の場合、**Max**(**x**) の出力は $z = 10$ となるような $[\![z]\!]^{\mathsf{A}}$ である。愚直に計算するなら 2 つの入力を大小比較するトーナメント方式が考えられるが、大小比較 **Comparison** の出力は論理シェアであるため、これを毎回算術シェアに変換して **MULT** によって値を取り出すとラウンド数が増えてしまう。そこで、まずは最大の要素が入っている箇所だけ 1 が立つようなベクトル（先程の **x** の例なら $[0, 1, 0, 0]$）を生成することを目標とする。これが生成できれば、これらを **B2A** で算術シェアに変換し、**x** との内積を計算すれば最大値を抜き出すことができる。

まず初めに、各要素を他の全ての要素と大小比較し、その結果の **AND** を計算する（この結果を **y** とする）。最大の要素は他のどの要素よりも大きいので大小比較の結果は全て 1 が立つはずであり、これで目標が達成できるように思われるが、ベクトル内に 1 位タイとなるような要素が 2 つ以上あった場合は複数箇所に 1 が立ってしまう[*7]。これを回避するために **y** に **MSNZB** を適用して 1 を 1 つに絞り、**B2A** 及び内積を計算する。最小値抽出プロトコルも大小比較プロトコルへの要素の入力順序を入れ替えることでほぼ同様に実現できる。ベクトルの要素数が 8 以下の場合、大小比較に 4 ラウンド、**AND** に 1 ラウンド、**MSNZB** に 1 ラウンド、**B2A** に 2 ラウンド、**DotProduct** に 1 ラウンドで、合計 9 ラウンドで計算できる。

3.2.7 最大値（最小値）インデックス抽出

最大値インデックス抽出プロトコル **Argmax** はベクトル内最大要素のインデックスを出力するプロトコルである。ほぼ **Max** と同様の構成であるが、**Argmax** の場合は最後の内積 **DotProduct** をパブリックなインデックスとの積に置き換えてよいため、1 ラウンド減らすことができる。

3.2.8 除算

除算プロトコル **Division**($[\![N]\!]^{\mathsf{A}}, [\![D]\!]^{\mathsf{A}}$) 分子 N と分母 D の算術シェアを入力し、（小数点以下を切り捨てた）商

[*7] この状態で内積を計算すると、（最大値）×（1 が立っている個数）が計算結果として出力されてしまう。

$[\![\frac{N}{D}]\!]^{\mathsf{A}}$ を出力するプロトコルである。構成は非常に複雑であり、具体的な構成は [4] などを参照されたい。**Overflow** や **RightShift** のラウンド数が効率化されたことで **Division** も自然に効率化されている。また、**Comparison** が 4 ラウンドで計算できるため、D の逆数を推定するフェーズを $i = 1, \cdots, n$ 全てに対する 2^{i-1} と D の大小比較操作で実現することによってラウンド数を削減できる。紙面の都合で詳細な説明は省略する。

4. 提案プロトコルの性能評価

本章では 3 章で提案したプロトコルの性能評価結果を示す。我々は提案したゲート・プロトコルのシミュレータを Python3.6(+Numpy v1.13.1) を用いて実装した。2 台（あるいは事前計算を行うマシンを含めて 3 台）をネットワークで接続して実験したわけではなく、一つのプログラムの上で事前計算及び 2 サーバの動作を記述し、通信は「行ったことにする（ラウンド数や通信ビット数は別途数える）」という実装なのでシミュレータと呼んでいる。実験には Core i7-6700K 4.00GHz、64GB RAM のマシンを用いた。n は $\{8, 16, 32, 64\}$ に対応しているが、実験結果は紙面の都合上 $n = 32$ のもののみを掲載する。また、バンド幅と遅延は任意の値を設定して計算できるが、[22], [23] などを参考に、LAN 環境としてバンド幅 1Gbps で通信遅延が 0.2ms、WAN 環境としてバンド幅 10Mbps で通信遅延が 40ms のネットワークを仮定して数値を計算した。

4.1 多入力論理積ゲートの性能評価

まず、N 入力論理積 **AND** ゲートの性能を評価した結果を示す。シミュレーションは 1 つの入力を処理する時間の測定と、100 入力同時（100 並列）に処理する時間の測定を両方行った。結果は図 1,2 の通りである。数字は計算時間のみで通信に関する時間は含んでいないことに注意されたい。結果から以下のことがわかる。

- 計算時間は WAN の通信遅延と比べると非常に小さい。2 入力 **AND** のみを用いて 8 入力の論理積を計算しようと思えばツリー状に配置しても 3 ラウンド（WAN なら遅延だけで 120ms）を要することを考えると、多入力 **AND** ゲートは非常に強力なツールである。
- 並列度を 100 倍に上げても（行列演算ライブラリを活用しているので）計算時間は 3〜5 倍程度にしかならない。一方で、事前計算はその大部分を乱数生成に費やしているため、計算時間が並列数に対して線形に増加する。

4.2 提案プロトコルの性能評価

3.2 節の手法でプロトコルを実装して性能評価を行った。こちらもゲート同様 1 並列と 100 並列で処理する時間の測定を行った。結果は表 1 の通りである。オンライン計算と比べて事前計算に時間がかかることや、並列数を上げてもオンライン計算の実行時間にはそれほど大きな影響が出ないことはゲートレベルでの評価と同様の結果である。WAN 環境の場合は 100 並列程度ではまだまだビット時間よりレイテンシの影響が大きい。除算以外のプロトコルはオンライン時間の 90%以上がレイテンシによるものであり、ラウンド数を削減することの意義が大きい。

5. 秘匿畳み込みニューラルネットワーク

5.1 必要な演算とその構成の概要

次のような状況を想定し、関数の構築やパラメータの設定を行った。

- MNIST をパディングした 30×30 の画像を入力として扱う。まず、ウインドウサイズ 4、ストライド 2 の畳み込みを行う。フィルタの枚数は 8 枚に設定した。畳み込みは内積演算によって処理可能であり、この時点で画像サイズは 14×14 になる。
- その後 2×2 の Max プーリングを行うが、これは **Max** プロトコルで実現できる。得られた 7×7 の画像に対して活性化関数 ReLU を適用する。ReLU は 0 と x と **Comparison** で大小比較した後、出力を **B2A** によって算術シェアに変換してから x を乗じればよい。
- もう一度畳み込みを行う。ウインドウサイズ 2、ストライド 1、フィルタ 8 枚に設定した。この処理によって画像サイズは 6×6 になっており、この画像に対して活性化関数 ReLU を適用する。
- 全てのフィルタの各値を一列に展開する。全結合層の大きさは 288×10 となり、内積演算があれば実現できる。推論だけであればここで計算は終了である。
- Softmax 関数と損失関数を計算する。指数関数 $\exp(x)$ は [26] の通り、3 次のマクローリン展開で近似することにし、除算はエラー訂正イテレーションを減らしても誤差はかなり小さいという結果 [27] に基づき、5 周のエラー訂正を 2 周に削減する。
- 逆伝搬は特に言及すべき点はないが、ReLU と Max プーリングは前方伝搬の際の途中計算結果を記憶しておくことで、乗算 1 回のみで処理できる。

また全体を通して、小数の演算は [23] に示されている通り、入力を 2^{ℓ} 倍してから計算し、乗算後には ℓ ビット右シフトするという処理で代替する。$n = 32$、ミニバッチサイズは 300 に設定し、学習率は公開値として扱う。

5.2 性能評価

まずは推論であるが、画像 300 枚の推論にオンライン計算時間が 137.9s、通信ラウンド数が 36、総通信量 389.1MB を要するという結果が出た。LAN 環境であれば 138.3s、WAN 環境でも 178.3s ほどで計算が完了するということに

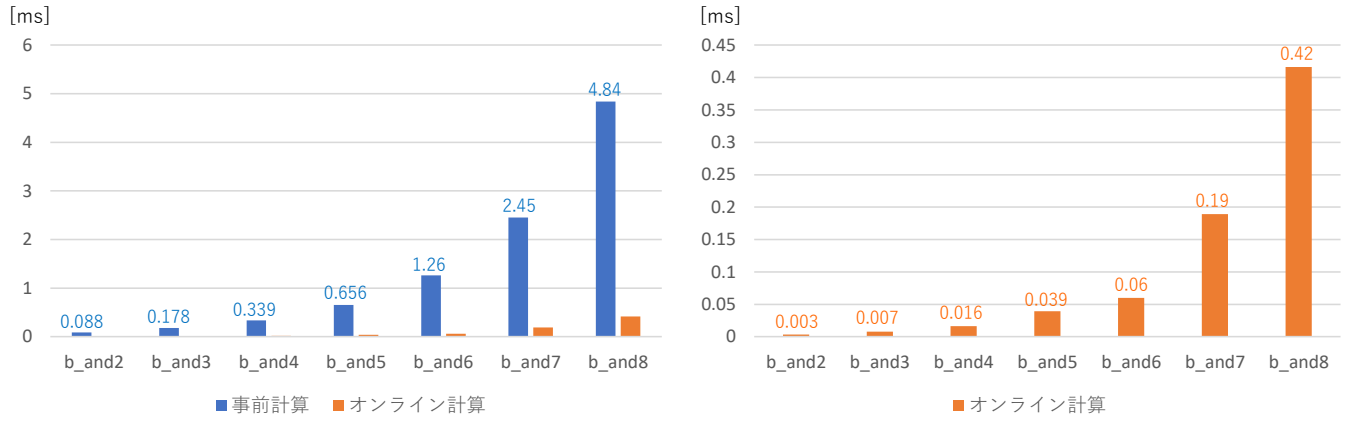

図 1 $n = 32$ における N 入力 **AND** ゲート（$2 \leq N \leq 8$）の計算時間：左のグラフが事前計算およびオンライン計算で、右のグラフはオンライン計算を抜き出したものである。

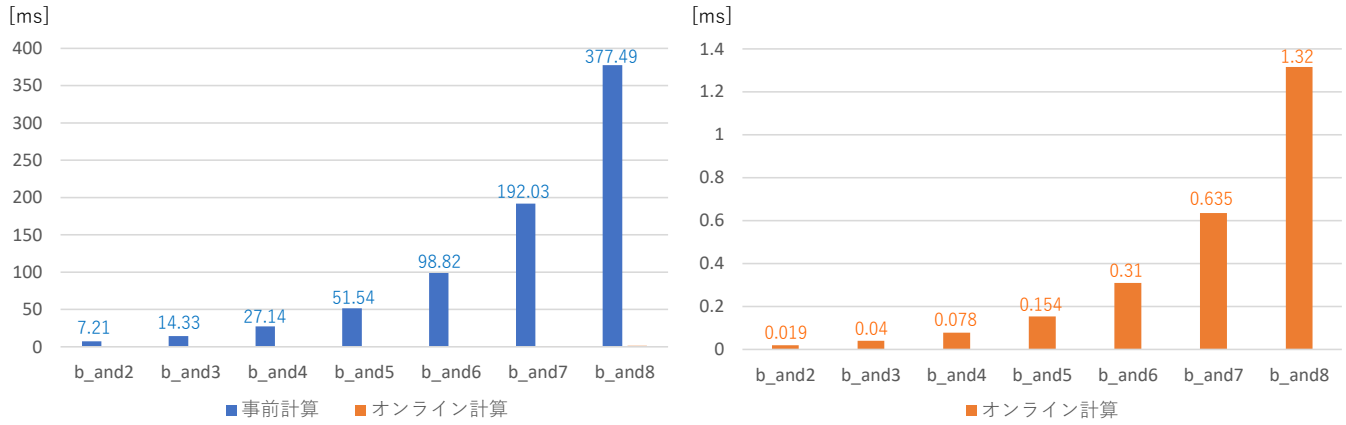

図 2 $n = 32$ における100 並列N 入力 **AND** ゲート（$2 \leq N \leq 8$）の計算時間：左のグラフが事前計算およびオンライン計算で、右のグラフはオンライン計算を抜き出したものである。

プロトコル名	オフライン計算	オンライン計算	LAN ビット時間	LAN レイテンシ	LAN オンライン合計	WAN ビット時間	WAN レイテンシ	WAN オンライン合計
Equality	1.47	0.55	–	0.4	0.95	–	80	80.55
	19.77	0.58	–	0.4	0.98	0.05	80	80.62
MSNZB	9.40	3.35	–	0.4	3.75	0.02	80	83.37
	129.2	3.27	0.02	0.4	3.69	2.10	80	85.37
Comparison	28.49	10.33	–	0.8	11.13	0.07	160	170.4
	409.8	10.54	0.07	0.8	11.41	6.68	160	177.2
RightShift	19.01	6.92	–	1.0	7.92	0.05	200	207.0
	273.4	7.46	0.05	1.0	8.50	4.58	200	212.0
3−Max	73.20	21.36	0.004	1.8	23.16	0.42	360	381.8
	2300.6	36.33	0.42	1.8	38.55	41.9	360	438.2
3−Argmax	73.17	21.58	0.004	1.6	23.18	0.41	320	342.0
	2299.9	36.47	0.41	1.6	38.48	41.2	320	397.7
Division	1667.4	621.6	0.04	12.0	633.6	3.89	2400	3025.5
	25058.3	1205.8	3.89	12.0	1221.7	388.9	2400	3994.7

表 1 $n = 32$ におけるプロトコルの性能評価結果：上段/下段はそれぞれ 1 並列/100 並列での数値である。シミュレータは事前計算を含む形で各プロトコルにおけるサーバ 2 台分の演算を行うため、まずは通常通りにプログラムを実行し、その実行にかかった時間から別途計測しておいた事前計算に要する時間を差し引いた後に 2 で割ってオンライン計算時間を算出した。ビット時間は（送信ビット数）÷（バンド幅）、レイテンシは（ラウンド数）× 通信遅延で算出している。単位は ms で、1μs より小さい場合は – を表示している。**Max** と **Argmax** はそれぞれ 3 要素の中から最大値/最大値インデックスを計算する場合の値を記載した。

なる。CNN は複数枚のフィルタを使うこともあり、一度に並列処理するデータ量が非常に多くなる。そのため通信遅延の影響は相対的に小さくなるが、事前計算量は非常に多い。今回のケースでも事前計算に 105 分以上かかってしまう。

訓練はというと、LAN/WAN を問わず、まだ現実的とは言えない。1 ラウンドの訓練に事前計算が 80 分以上必要である。10 エポック訓練するためには 2000 ラウンドの計算が必要であり、これは 2600 時間以上（111 日以上）になる。Softmax 関数における除算が重いことの他に、整数を小数に変換する作業（**RightShift**）を逆伝搬で膨らんだデータに対して行う操作も非常に高コストであることがわ

かった。

6. まとめと今後の課題

本稿では秘密分散ベースの 2 者計算に注目し、ラウンド効率のよいプロトコルの提案とシミュレータによる性能評価を行った。また、まだ実現が難しいと思われる畳み込みニューラルネットワーク向けの関数を適当なパラメータで動かし、今後の課題を探った。事前計算の高速化が最も重要な課題で、プロトコルレベル/実装レベルの両面から解を探っていく必要がある。特に、今回の実装ではハードウェア（具体的には AES-NI）による乱数生成の高速化を採用していないので、これを取り入れるだけで事前計算速度は大幅に向上することが期待される。また、「(3 者計算ではすでに存在 [22] している）効率的な小数演算」や「除算の効率化」、「秘匿計算フレンドリなネットワーク構造や訓練手法の構築」も興味深い課題である。

謝辞 本研究の一部は JST CREST JPMJCR1688 の支援を受けて行われた。

参考文献

[1] T. Araki, J. Furukawa, Y. Lindell, A. Nof, K. Ohara. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. *CCS 2016*, pp.805–817, 2016.

[2] T. Araki, A. Barak, J. Furukawa, T. Lichter, Y. Lindell, A. Nof, K. Ohara, A. Watzman, O. Weinstein. Optimized Honest-Majority MPC for Malicious Adversaries - Breaking the 1 Billion-Gate Per Second Barrier. *S&P 2017*, pp.843–862, 2016.

[3] D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. *CRYPTO 1991*, LNCS 576, pp.420–432, 1991.

[4] D. Bogdanov, M. Niitsoo, T. Toft, J. Willemson. High-Performance Secure Multi-Party Computation for Data Mining Applications. *Int. J. Inf. Sec.*, 11(6), pp.403–418, 2012.

[5] R. Bost, R.A. Popa, S. Tu, S. Goldwasser. Machine Learning Classification over Encrypted Data. *NDSS 2015*, 2015.

[6] F. Bourse, M. Minelli, M. Miniholdm, P. Paillier. Fast Homomorphic Evaluation of Deep Discretized Neural Networks. *CRYPTO(3) 2018*, LNCS 10993, pp.483–512, 2018.

[7] H. Chabanne, A. Wargny, J. Milgram, C. Morel, .E Prouff. Privacy-Preserving Classification on Deep Neural Network. *RWC 2017 (ePrint Archive 2017/035)*, 2017.

[8] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, A. Nof. Fast Large-Scale Honest-Majority MPC for Malicious Adversaries. *CRYPTO(3) 2018*, LNCS 10993, pp.34–64, 2018.

[9] I. Damgård, M. Fitzi, E. Kiltz, J. Nielsen, T. Toft. Unconditionally Secure Constant-Rounds Multi-Party Computation for Equality, Comparison, Bits and Exponentiation. *TCC 2006*, pp. 285–304, 2006.

[10] D. Demmler, T. Schneider, M. Zohner. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. *NDSS 2015*, 2015.

[11] K. Fukushima, S. Miyake. Neocognitron: A New Algorithm for Pattern Recognition Tolerant of Deformations and Shifts in Position. *Pattern Recognition, 15(6):455–469*, 1982.

[12] 古川 潤. スループットの大きいマルチパーティー計算. *SCIS 2016*, 2016.

[13] R. Gilad-Bachrach, N. Dowlin, K. Laine, K.E. Lauter, M. Naehrig, J. Wernsing. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. *ICML 2016*, pp.201–210, 2016.

[14] O. Goldreich. The Foundations of Cryptography - Volume 2, Basic Applications. *Cambridge University Press*, 2004.

[15] O. Goldreich, S. Micali, A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. *STOC 1987*, pp.218–229, 1987.

[16] T. Graepel, K.E. Lauter, M. Naehrig. ML Confidential: Machine Learning on Encrypted Data. *ICISC 2012*, LNCS 7839, pp.1–21, 2013.

[17] E. Hesamifard, H. Takabi, M. Ghasemi, R.N. Wright. Privacy-preserving Machine Learning as a Service. *PoPETs 2018(3)*, pp.123–142, 2018.

[18] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proc. of the IEEE*, pp.2278–2324, 1998.

[19] J. Liu, M. Juuti, Y. Lu, N. Asokan. Oblivious Neural Network Predictions via MiniONN Transformations. *CCS 2017*, pp.619–631, 2017.

[20] W. Lu, J. Sakuma. Crypt-CNN(II): Crytographically Evaluate Non-linear Convolutional Neural Network. *CSS 2017*, pp.773–780, 2017.

[21] P. Mohassel, O. Orobets, B. Riva. Efficient Server-Aided 2PC for Mobile Phones. *PoPETs 2016(2)*, pp.82–99, 2016.

[22] P. Mohassel, P. Rindal. ABY3: A Mixed Protocol Framework for Machine Learning. *CCS 2018*, to appear.

[23] P. Mohassel, Y. Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. *S&P 2017*, pp.19–38, 2017.

[24] H. Morita, N. Attrapadung, T. Teruya, S. Ohata, K. Nuida, G. Hanaoka. Constant-Round Client-Aided Secure Comparison Protocol. *ESORICS(2) 2018*, LNCS 11099, pp.395–415, 2018.

[25] V. Nair, G.E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *ICML 2010*, pp.807–814, 2010.

[26] 大畑 幸矢. 秘匿深層学習再考. *SCIS 2018*, 2018.

[27] S. Ohata, H. Morita, G. Hanaoka. Accuracy/Efficiency Trade-Off for Privacy-Preserving Division Protocol. *ISITA 2018*, to appear.

[28] M.S. Riazi, C. Weinert, O. Tkachenko, E.M. Songhori, T. Schneider, F. Koushanfar. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. *AsiaCCS 2018*, pp.707–721, 2018.

[29] B.D. Rouhani, M.S. Riazi, F. Koushanfar. DeepSecure: Scalable Provably-Secure Deep Learning. *ePrint Archive 2017/502*, 2017.

[30] S. Siim. A Comprehensive Protocol Suite for Secure Two-Party Computation. *Master Thesis, University of Tartu*, 2016.

[31] S. Wagh, D. Gupta, N. Chandran. SecureNN: Efficient and Private Neural Network Training. *ePrint Archive 2018/442*, 2018.

[32] A.C. Yao. How to Generate and Exchange Secrets (Extended Abstract). *FOCS 1986*, pp.162–167, 1986.