Oblivious Polynomial Evaluation and Oblivious Neural Learning*

Yan-Cheng Chang† Chi-Jen Lu‡

Abstract

We study the problem of Oblivious Polynomial Evaluation (OPE). There are two parties, Alice who has a polynomial P, and Bob who has an input x. The goal is for Bob to compute $P(x)$ in such way that Alice learns nothing about x and Bob learns only what can be inferred from $P(x)$. Previously existing protocols were based on some newly-invented intractability assumptions that have not been well studied, so one may have doubts about the security of these protocols. In this paper, we propose OPE protocols which are only based on the standard primitive Oblivious Transfer, and still our protocols are more efficient in several natural cases. Our protocols can also be easily modified to handle multi-variate polynomials and polynomials over floating-point numbers. As an application, we study the problem of Oblivious Neural Learning, where one party has a neural network and the other, with some training set, wants to train the neural network in an oblivious way. We provide a protocol for this problem, which is based on our protocol for OPE.

Keywords: Cryptography; Security; Oblivious Polynomial Evaluation; Oblivious Transfer; Neural Learning.

Corresponding Author:

Chi-Jen Lu
Institute of Information Science
Academia Sinica
Taipei, Taiwan
Email: cjlu@iis.sinica.edu.tw
Phone: +886 2 27883799 ext. 1820
Fax: +886 2 27824814

*A preliminary version of this paper appeared in *Advances in Cryptology - ASIACRYPT'01*, pp. 369-384, 2001.

†The work was done while the author was in the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan. Current Address: Division of Engineering and Applied Sciences, Harvard University, USA. Email: ycchang@eecs.harvard.edu

‡Institute of Information Science, Academia Sinica, Taipei, Taiwan. Email: cjlu@iis.sinica.edu.tw

1 Introduction

Assume that there are two parties, Alice who has a function f and Bob who has an input x. They want to collaborate in a way for Bob to compute $f(x)$ such that Alice learns nothing about x and Bob learns only what can be inferred from $f(x)$. A protocol achieving this task for any function f and any input x is called an Oblivious Function Evaluation protocol. The remarkable results of Yao [17], Goldreich, Micali, and Wigderson [9], and Kilian [13] showed that such protocols do exist, under some standard cryptographic assumptions. Their protocols use a Boolean circuit to represent the function f and then simulate the computation of this circuit in some oblivious way. The computational or communicational overhead of their protocols depends only linearly on the circuit size of the function f, which is the best one can expect from a complexity-theoretical point of view. However, their protocols are far from being practical in general, and a lot of work still needs to be done. One line of research is to study cases when different representations of functions can lead to more efficient simulation.

Naor and Pinkas [15] considered polynomials over finite fields. Note that any function from m bits to m bits can be represented by a polynomial over a finite field $GF(2^m)$, but its degree could go as high as $2^m - 1$. So one would like to focus on those functions that can be represented by low degree polynomials. This turns out to have several interesting applications [15, 8, 14, 12]. The scheme proposed in [15] is much more efficient than the conventional way of going through oblivious circuit evaluation, but its security is based on two assumptions. One assumption is the existence of a secure Oblivious Transfer protocol while the other, proposed by themselves, is the intractability of a Noisy Polynomial Interpolation Problem. Bleichenbacher and Nguyen [3] later showed that this new assumption may be much weaker than expected and suggested the use of a possibly stronger intractability assumption on a Polynomial Reconstruction Problem. Still, no one can say how hard this problem is as it is not that well-studied. Recently, Lindell and Pinkas [14] mentioned a not-yet-published OPE protocol, which is also based on some newly proposed assumption. The assumption is that the Decisional Diffie-Hellman Assumption, denoted as DDH, also holds over the group $\mathbb{Z}^*_{n^2}$, where n is the product of two large primes. Contrary to the well studied DDH over $\mathbb{Z}^*_n$ [2], more research may need to be done before one can have some confidence on this new assumption. As there may be doubt on the security of both existing OPE protocols, a more satisfactory solution is certainly welcome.

As in [15, 14], we will focus on the case with semi-honest parties, who may be curious but still follow the protocol. The malicious case can be handled in some standard way using commitments and zero-knowledge proofs, which will only be briefly mentioned. We will propose three OPE protocols of different flavors. Compared to previous ones, the security of our first two protocols is only based on a well-accepted cryptographic assumption, namely, the existence of a secure 1-out-of-2 oblivious transfer protocol, denoted as OT^2_1. For polynomials of degree d over a finite field $\mathbb{F}$, our first protocol uses $d \log |\mathbb{F}|$ invocations of OT^2_1 while [15] needs $(2kd+1)\log m$ invocations of OT^2_1 for some unspecified integers k and $m \gg d$ depending on their proposed assumption.[1] Note that for the problem in their assumption to be intractable, at least m must be very large just to prevent a brute-force algorithm that tries every possibility. So, even with their additional security concern, their protocol is better than ours only when $|\mathbb{F}| > m^{2k}$, i.e. when $|\mathbb{F}|$ is very large. Moreover, other than carrying out OT's, our protocol involves only extremely simple computation. Our second protocol is less efficient than our first one, but we include it here as the technique for achieving security seems interesting and may have other applications. Our third protocol involves a third party who does not collude with others but may be curious, and our protocol is perfectly secure, without any cryptographic assumption. Unlike that of [15], all our protocols can immediately handle multi-variate polynomials.

Many important applications in real life involve numerical computation over floating-point numbers, instead of over integers or arbitrary finite fields. To do OPE over floating-point numbers, the approach of Lindell and Pinkas [14] is to embed floating-point numbers into a finite field, apply some OPE protocol over the finite field, and then convert the result back to a floating-point number. On the other hand, as our protocols evaluate polynomials in a more natural way, they can be easily modified to operate directly on floating-point numbers.

[1] Actually they use $2kd+1$ invocations of 1-out-of-m oblivious transfer, denoted as OT^m_1. It is known that one OT^m_1 can be simulated by $\log m$ calls to OT^2_1, together with several evaluations of a pseudo-random function [15].

In addition to computing functions obliviously, some computational tasks may also involve security issues and people may want to perform them in some oblivious way. We use machine learning as an example, and demonstrate the applicability of our OPE protocol over floating-point numbers. Lindell and Pinkas [14] considered the scenario where two parties, each holding a private database, want to jointly construct a decision tree that classifies entries in both databases, using a so-called ID3 algorithm. Such kind of learning is not robust to changes in the sense that small changes to a database may require the whole process to be run again. We use neural network as our learning model and consider the following scenario. Alice has a neural network which is trained to some degree and she uses it to serve the classification requests from other parties. Alice wants to keep her neural network secret, while others want to keep their requests secret. This is the task of oblivious neural computing. At some point, another party Bob with a set of training examples wants to help Alice's neural network get better, maybe for his own good later. Alice wants to have a secure learning process so that Bob learns nothing from her, while Bob also wants to keep his training set secret. Later, other parties having their own training set can help Alice too, and Alice's neural network can adapt in an incremental way. This is the task of oblivious neural learning. We will apply our OPE protocol over floating-point numbers, and derive protocols for oblivious neural computing and oblivious neural learning.

The rest of the paper is organized as follows. In Section 2, we give definitions and tools that will be used later. Three OPE protocols are proposed in Section 3. In Section 4, we show oblivious protocols for neural computing and learning.

2 Preliminaries

For a positive integer n, let $[n]$ denote the set $\{1,\ldots,n\}$. For an n-dimensional vector v, let v_i, for $i \in [n]$, denote the component in the i'th dimension, and we write $v = (v_1,\ldots,v_n) = (v_i)_{i\in[n]}$. Fix a security parameter τ, so that numbers about $2^{-\tau}$ are considered *negligible* and circuits of sizes about 2^{τ} are considered *infeasible*. For a distribution D over a set S, let $D(i)$, for $i \in S$, denote the probability of i according to D, and define $D(A)$, for $A \subseteq S$, to be $\sum_{i\in A} D(i)$.

Definition 1. *Let D and D' be two distributions over a set S. Their distance is defined as $d(D,D') = \max_{A\subseteq S} d_A(D,D')$, with $d_A(D,D') = |D(A) - D'(A)|$.*

Note that $d(D,D') = \frac{1}{2}\sum_{i\in S}|D(i) - D'(i)|$, which is a useful way for calculating $d(D,D')$.

Definition 2. *Let D and D' be two distributions. They are statistically indistinguishable, denoted as $D \overset{s}{\equiv} D'$, if $d(D,D')$ is negligible. They are computationally indistinguishable, denoted as $D \overset{c}{\equiv} D'$, if $d_A(D,D')$ is negligible for any subset A decided by a circuit of feasible size.*[2]

We will assume that parties in our protocols have only circuits of feasible sizes for computation unless mentioned otherwise. So we will focus on computational security, and the default distinguishability will be the computational one.

An important cryptographic primitive is the 1-out-of-2 oblivious transfer, denoted as OT_1^2. There are several variants which are all equivalent, and the one most suited for us is the following *string* version of OT_1^2. Let $\mathbb{F}$ be a set.

Definition 3. *An OT_1^2 protocol has two parties,* Sender *who has input $(x_0,x_1) \in \mathbb{F}^2$ and* Chooser *who has a choice $c \in \{0,1\}$. The protocol is* correct *if the Chooser learns x_c for any (x_0,x_1) and c. The protocol is* secure *if both conditions below are satisfied for any (x_0,x_1) and c:*

- *Chooser cannot distinguish the distribution of Sender's messages from that induced by Sender having a different value of x_{1-c}.*
- *Sender cannot distinguish the distributions of Chooser's messages induced by c and $1-c$.*

[2]Note that for A decided by a circuit C, $d_A(D,D') = |\Pr_{x\in D}[C(x)=1] - \Pr_{x\in D'}[C(x)=1]|$.

Similarly one can define OT_1^k for any $k \geq 3$, with Sender having k elements and Chooser wanting to learn one. We will use OT_1^k, for $k \geq 2$, to denote an assumed correct and secure OT_1^k protocol. It is known that the existence of OT_1^2 implies the existence of OT_1^k for any $k \geq 3$ [5, 15].

Definition 4. *A protocol for* oblivious polynomial evaluation *has two parties, Alice who has a polynomial P over some finite field $\mathbb{F}$ and Bob who has an input $x_* \in \mathbb{F}$. An OPE protocol is* correct *if Bob learns $P(x_*)$ for any x_* and P. It is* secure *if both conditions below are satisfied for any x_* and P:*

- *Alice cannot distinguish the distribution of Bob's messages from that induced by Bob having a different x'_*.*
- *Bob cannot distinguish the distribution of Alice's messages from that induced by Alice having a different P' with $P'(x_*) = P(x_*)$.*

We say that a party in a protocol is *semi-honest* if the party follows the protocol but may try to learn more information than he or she should. We only focus on semi-honest parties in this paper. The case of *malicious* parties can be handled in a standard way, using commitments and zero-knowledge proofs, which will only be briefly sketched for our first protocol.

Suppose D and D' are two distributions depending on distributions E and E' respectively. For any possible outcome t of E and E', let $(D|E = t)$ and $(D'|E' = t)$ denote the distributions of D and D' conditioned on $E = t$ and $E' = t$ respectively. Here is a useful lemma for showing $D \overset{c}{\equiv} D'$, which will be used several times in our security proofs later.

Lemma 1. *$D \overset{c}{\equiv} D'$ provided $E \overset{s}{\equiv} E'$ and $(D|E = t) \overset{c}{\equiv} (D'|E' = t)$ for any t.*

Proof. Let C be a circuit which outputs 1 with probabilities p and p' with respect to D and D'. Let p_t and p'_t denote the corresponding probabilities with respect to $(D|E = t)$ and $(D'|E' = t)$. Let $q_t = E(t)$ and $q'_t = E'(t)$. Then

$$\begin{aligned} |p - p'| &= \left|\sum_t q_t p_t - \sum_t q'_t p'_t\right| \\ &\leq \sum_t |q_t p_t - q_t p'_t| + \sum_t |q_t p'_t - q'_t p'_t| \\ &\leq \sum_t q_t |p_t - p'_t| + \sum_t |q_t - q'_t| \end{aligned}$$

So if $\sum_t |q_t - q'_t|$ is negligible and each $|p_t - p'_t|$ is negligible, then $|p - p'|$ is negligible. □

Some cases later have identical E and E', and we only need to check each $|p_t - p'_t|$.

A family H of functions from S_1 to S_2 is said to satisfy a *pair-wise independent property* if for any distinct $\alpha, \alpha' \in S_1$,

$$\Pr_{h \in H}[h(\alpha) = h(\alpha')] = \frac{1}{|S_2|}.$$

Let $(H, H(S_1))$ denote the distribution of $(h, h(v))$ with random $h \in H$ and random $v \in S_1$, and let (H, S_2) denote the uniform distribution over $H \times S_2$. We will use the following lemma, which is a special case of the so-called Leftover Hash Lemma [10, 11].

Lemma 2. *Let H be any family of functions from S_1 to S_2 satisfying the pair-wise independent property. Then $d((H, H(S_1)), (H, S_2)) \leq \frac{1}{2}\sqrt{\frac{|S_2|}{|S_1|}}$.*

A proof of this lemma is given in the appendix for completeness.

3 Oblivious Polynomial Evaluation Protocols

We will present three OPE protocols of different flavors in this section. Assume that both parties have agreed that polynomials are over a finite field $\mathbb{F}$ and have degrees at most d. The set of such polynomials can be identified with the set $\mathbb{T} = \mathbb{F}^{d+1}$ in a natural way. Suppose now Alice has a polynomial $P(x) = \sum_{i=0}^{d} a_i x^i \in \mathbb{T}$ and Bob has $x_* \in \mathbb{F}$.

3.1 The First Protocol for OPE

To make the picture clear, we only discuss the case $\mathbb{F} = GF(p)$ for some prime p. The generalization to $GF(p^k)$ with $k > 1$ is straightforward. Let $m = \lceil \log_2 |\mathbb{F}| \rceil$. Each coefficient a_i in the polynomial can be represented as $a_i = \sum_{j \in [m]} a_{ij} 2^{j-1}$ with $a_{ij} \in \{0, 1\}$. For $i \in [d]$ and $j \in [m]$, let $v_{ij} = 2^{j-1} x_*^i$. Note that for each $i \in [d]$, $\sum_{j \in [m]} a_{ij} v_{ij} = a_i x_*^i$. The idea is to have Bob prepare $(v_{ij})_{j \in [m]}$ and have Alice get those v_{ij} with $a_{ij} = 1$, in some secret way. This is achieved by having Bob prepare the pair $(r_{ij}, v_{ij} + r_{ij})$ for a random noise r_{ij}, and having Alice get what she wants via OT_1^2. Note that what Alice obtains is $a_{ij} v_{ij} + r_{ij}$. Here is our first protocol, basing only on the existence of secure OT_1^2.

Protocol 1

1. Bob prepares dm pairs $(r_{ij}, v_{ij} + r_{ij})_{i \in [d], j \in [m]}$, with each r_{ij} chosen randomly from $\mathbb{F}$.
2. For each pair $(r_{ij}, v_{ij} + r_{ij})$, Alice runs an independent OT_1^2 with Bob to get r_{ij} if $a_{ij} = 0$ and $v_{ij} + r_{ij}$ otherwise.
3. Alice sends to Bob the sum of a_0 and those dm values she got. Bob subtracts $\sum_{i,j} r_{ij}$ from it to obtain $P(x_*)$.

Lemma 3. *Protocol 1 is correct when parties are semi-honest.*

Proof. The sum Bob obtains in Step 3 is $a_0 + \sum_i \sum_j (a_{ij} v_{ij} + r_{ij}) = P(x_*) + \sum_{i,j} r_{ij}$. □

Lemma 4. *Protocol 1 is secure when parties are semi-honest.*

Proof. First, we prove Alice's security. Suppose P and P' are two distinct polynomials with $P(x_*) = P'(x_*) = y_*$. According to Lemma 1, it suffices to show that for any fixed $(r_{ij})_{i \in [d], j \in [m]}$, Alice's respective message distributions D and D' induced by P and P' are indistinguishable. Note that the last message from Alice is $y_* + \sum_{i,j} r_{ij}$ for both P and P' and can be ignored. So we focus on Alice's dm messages from the dm independent executions of OT's. For $0 \le k \le dm$, let D_k denote the distribution with the first k messages from D and the remaining messages from D'. Assume that there exists a distinguisher C for D and D'. A standard argument shows that C can also distinguish $D_{k_0 - 1}$ and D_{k_0} for some k_0. Note that Alice must select different elements from that pair in the k_0'th OT, as otherwise the two distributions are identical. Then one can break Chooser's security in OT_1^2 when Sender has this input, because with Chooser's messages for different choices replacing the k_0'th message of D_{k_0-1}, we get exactly D_{k_0-1} and D_{k_0}, which can be distinguished by C. As OT_1^2 is assumed to be secure, D and D' are indistinguishable, and Alice is secure.

Next, we prove Bob's security. Note that Bob sends dm messages to Alice for the dm independent executions of OT's. Let $x_* \neq x'_*$, let E and E' be Bob's respective message distributions, and let E_k denote the distribution with the first k messages from E and the remaining messages from E'. Suppose a distinguisher for E and E' exists. Then it can also distinguish E_{k_0-1} and E_{k_0} for some k_0. The pairs in that k_0'th OT have the forms $(r, v + r)$ and $(r', v' + r')$, for some fixed v and v' and for random r and r'. Alice's polynomial is fixed, so which element to choose in that k_0'th OT is also fixed. Suppose Alice chooses the first one in that pair. Then according to Lemma 1, there is a fixed r_0 such that E_{k_0-1} conditioned on Bob having $(r_0, v + r_0)$ and E_{k_0} conditioned on Bob having $(r_0, v' + r_0)$ are distinguishable. Similarly as before, one can distinguish Sender's messages when Sender has $(r_0, v+r_0)$ and $(r_0, v'+r_0)$ respectively and Chooser selects the first element, which violates Sender's security in OT_1^2. The case when Alice chooses the second one in that pair can be argued similarly, by noticing that the distribution $(r, v+r)$ and the distribution $(-v+r, r)$ are identical. As OT_1^2 is assumed to be secure, so is Bob. □

Theorem 5. *Protocol 1 is correct and secure when parties are semi-honest.*

Note that only dm invocations of OT_1^2 are required and they can be done concurrently. Also observe that if OT_1^2 can achieve perfect security for Chooser (e.g. [1]) in the information-theoretical sense, then so is Protocol 1 for Alice.

A slight modification to Protocol 1 can handle the case of malicious parties. The only complication is to enforce a malicious Bob to prepare dm pairs that are *consistent* in the sense that there is some x_* such that $v_{ij} = 2^{j-1}x_*^i$ for every i and j, which can be achieved as follows. Bob sends his commitments of dm pairs to Alice, Alice uses OT_1^2 to have her dm choices decommitted, and Bob uses a zero-knowledge proof to convince Alice that those dm pairs are consistent. All these can be done using, for example, the methods in [13].

3.2 The Second Protocol for OPE

The idea of our second protocol is to have Alice hide the random shares of her polynomial P among other random polynomials, have Bob evaluate all of them on his input x_*, and then have Alice select those values corresponding to the shares, which sum to $P(x_*)$. Recall that $\mathbb{T} = \mathbb{F}^{d+1}$. Let $n = \log|\mathbb{T}| + 2\tau$. For $P \in \mathbb{T}$ and $R = (R_1, \ldots, R_n) \in \mathbb{T}^n$, define the function $h_{R,P} : \{0,1\}^n \to \mathbb{T}$ as

$$h_{R,P}(\alpha) = P - \sum_{i\in[n]} \alpha_i R_i.$$

It's easy to check that for any $P \in \mathbb{T}$, the class $H_P = \{h_{R,P} : R \in \mathbb{T}^n\}$ satisfies the pair-wise independent property. Here is our second OPE protocol, which is also based on OT_1^2 only.

Protocol 2

1. Alice generates random $R \in \mathbb{T}^n$ and $\alpha \in \{0,1\}^n$ and sends $(R_1, \ldots, R_n, h_{R,P}(\alpha))$ to Bob. Let $R_{n+1} = h_{R,P}(\alpha)$ and $\alpha_{n+1} = 1$.
2. Bob generates random $r \in \mathbb{F}^{n+1}$ and prepares $n+1$ pairs $(r_i, R_i(x_*) + r_i)_{i\in[n+1]}$.
3. For pair i, Alice runs an OT_1^2 with Bob to get r_i if $\alpha_i = 0$ and $R_i(x_*) + r_i$ otherwise.
4. Alice sends the sum of the $n+1$ values to Bob. Bob subtracts $\sum_{i=1}^{n+1} r_i$ from it to get $P(x_*)$.

Theorem 6. *Protocol 2 is correct and secure when parties are semi-honest.*

Proof. The correctness is obvious because the sum that Bob obtains in Step 4 is $\sum_{i=1}^{n+1} \alpha_i R_i(x_*) + \sum_{i=1}^{n+1} r_i = P(x_*) + \sum_{i=1}^{n+1} r_i$. Bob's security proof is almost identical to that of Protocol 1, so we only prove Alice's security here.

Fix any two polynomials $P, P' \in \mathbb{T}$, let D and D' denote Alice's respective message distributions, and let E and E' be Alice's respective message distributions in Step 1. According to Lemma 1, it suffices to show $E \stackrel{s}{\equiv} E'$ and $(D|E=t) \stackrel{c}{\equiv} (D'|E'=t)$ for each $t \in \mathbb{T}$. Using an argument similar to that in Protocol 1, one can show $(D|E=t) \stackrel{c}{\equiv} (D'|E'=t)$ for each $t \in \mathbb{T}$ as otherwise one can break Chooser's security in OT_1^2. Note that the family H_P satisfies the pair-wise independent property and E is the distribution $(H_P, H_P(\{0,1\}^n))$. With $n = \log|\mathbb{T}| + 2\tau = (d+1)m + 2\tau$, Leftover Hash Lemma (Lemma 2) guarantees that the distance between E and the uniform distribution is at most $\sqrt{|\mathbb{T}|2^{-n}} = 2^{-\tau}$, which is negligible. Similarly E' also has a negligible distance to the uniform one. So $d(E, E')$ is negligible and $E \overset{s}{\equiv} E'$. According to Lemma 1, Alice is secure. $\square$

Note that there are $(n+1)\log|\mathbb{T}| = O(dm(dm+\tau))$ bits sent in Step 1, $O(dm+\tau)$ executions of OT_1^2 in Step 3, and m bits sent in Step 4.

3.3 A Protocol for 3-Party OPE

Here we show how to remove the use of OT_1^2 with the help a third party Clark. As a result, our protocol does not rely on any cryptographic assumption and is information-theoretically secure when no collusion exists. Again, we assume that Alice has a polynomial $P \in \mathbb{T}$, Bob has $x_* \in \mathbb{F}$ and only Bob learns $P(x_*)$. Now the security must also hold against Clark so that the messages he receives altogether look completely random to him; i.e.,

- *Clark cannot distinguish the uniform distribution from the joint distribution of messages he receives from Alice and Bob.*

Note that our model is slightly different from that of Feige, Kilian, and Naor [7], who have Clark as the party to receive the result. Here is the protocol.

Protocol 3

1. Bob sends random $(r_i)_{i\in[k]} \in \mathbb{F}^k$ to Alice. He also sends $(x'_i = x_*^i + r_i)_{i\in[k]}$ to Clark.
2. Alice sends random $(s_i)_{0\le i\le k} \in \mathbb{F}^{k+1}$ to Bob. She also sends $a'_0 = a_0 + s_0 - \sum_{i\in[k]} a_i r_i$ and $(a'_i = a_i + s_i)_{i\in[k]}$ to Clark.
3. Clark sends $y = a'_0 + \sum_{i\in[k]} a'_i x'_i$ to Bob, and Bob gets $P(x_*) = y - (s_0 + \sum_{i\in[k]} x'_i s_i)$.

Theorem 7. *Protocol 3 is correct and perfectly secure provided no collusion exists,*

Proof. The correctness is easy to verify. What Alice or Clark receives is completely random. Bob receives random $(s_i)_{0\le i\le k}$ in Step 2, and receives $P(x_*) + s_0 + \sum_{i\in[k]}(x_*^i + r_i)s_i$ in Step 3, so he sees the same distribution for any polynomial P' with $P'(x_*) = P(x_*)$. So each party is perfectly secure as long as no collusion exists. $\square$

3.4 Generalizations

It is not hard to see that all the protocols in this section can be easily extended to deal with multi-variate polynomials. In particular, we can solve an interesting special case: Alice has $a = (a_i)_{i\in[n]} \in \mathbb{F}^n$ while Bob has $x = (x_i)_{i\in[n]} \in \mathbb{F}^n$ and wants to learn the inner product $a \cdot x = \sum_{i\in[n]} a_i x_i$.

We have only considered the setting where Alice and Bob have their own inputs and Bob gets the final result. Later we will see a variation with each input and output shared by the two parties. We call this *computing with random shares*. Let's use the inner product function as an example. Suppose that Alice has $u, v \in \mathbb{F}^n$ and Bob has $u', v' \in \mathbb{F}^n$. They want to compute the inner product of $u+u'$ and $v+v'$, and produce random shares, one for each party, that sum to the inner product. This generalization can be reduced to the original problem in the following way. Note that $(u+u')\cdot(v+v')$ is equal to

$$(u\cdot v) + (u\cdot v' + v\cdot u') + (u'\cdot v').$$

Now Alice generates a random $r \in \mathbb{F}$ and prepares the $2(n+1)$-dimensional vector

$$a = (-r + u\cdot v, u_1, \dots, u_n, v_1, \dots, v_n, 1),$$

while Bob prepares the $2(n+1)$-dimensional vector

$$x = (1, v'_1, \ldots, v'_n, u'_1, \ldots, u'_n, u' \cdot v').$$

Bob can obtain $a \cdot x = -r + (u+u') \cdot (v+v')$ using a protocol for the original problem, and each party now holds a random share of the inner product $(u+u') \cdot (v+v')$. The variation for multi-variate polynomials can be handled similarly.

4 Oblivious Neural Learning

In this section, we introduce the problem of Oblivious Neural Learning, and then give a protocol for this problem. As we deal with floating-point numbers in this setting, we first show how to modify our protocols in the previous section to obtain protocols for floating-point numbers.

4.1 OPE Protocols for Floating-Point Numbers

We first give the definition of a floating-point number system.

Definition 5. *A floating-point number is a rational number $b = \pm \sum_{j=1}^{2m} b_j 2^{m-j}$ for some m, with $b_j \in \{0,1\}$. Let $\hat{m}$ denote the floating-point number system containing all such numbers together with standard arithmetic operations.*

Such a floating-point number can be represented by $2m+1$ bits: m bits for the fractional part, m bits for the integral part, and 1 bit for the sign. Unlike finite fields, operations in a floating-point number system are not closed and errors may occur because of the limitation of finite precision. An *underflow* occurs when the produced number needs more bits for the fractional part, and a *rounding* takes place to convert it into the nearest number in the floating-point number system. An *overflow* occurs when the produced number needs more bits for the integral part, and the result is left undefined.

When we want to hide an element v of a finite field $\mathbb{F}$ in our previous protocols, we generate a pair $(r, r+v)$ with a random $r \in \mathbb{F}$, so that any element of the pair itself looks completely random. There is a slight complication for floating-point numbers, but it can be easily fixed.

Lemma 8. *Suppose $v, v' \in \hat{\ell}$ for some ℓ and suppose $k \geq \ell + \tau + 1$. The distributions of $v + r$ and $v' + r'$ with random $r, r' \in \hat{k}$ have a negligible distance.*

Proof. The distance is at most $\frac{|v-v'|}{2(2^k - 2^{-k})+1} \leq \frac{2^{\ell+1}}{2^k} \leq 2^{-\tau}$. □

Next, we describe how to do OPE over floating-point numbers. Assume Alice holds $P(x) = \sum_{i=0}^{d} a_i x^i$, where $a_i \in \hat{m}$, and Bob holds $x_* \in \hat{m}$. For each i, let $|a_i| = \sum_{j=1}^{2m} a_{ij} 2^{m-j}$, with $a_{ij} \in \{0,1\}$. All our previous protocols can be easily modified for floating-point numbers, and here we only demonstrate one, which comes from Protocol 1. We will use OT_1^3, which can be implemented by 2 executions of OT_1^2 [15]. Let $k = (d+1)m + \tau + 1$ and $n = k + \log(2dm)$. Parties agree on the floating-point number system $\hat{k}$ for random numbers, and the floating-point number system $\hat{n}$ for all arithmetics so that no underflow or overflow will ever occur. Let $v_{ij} = 2^{m-j} x_*^i$.

Protocol 4

1. Bob prepares $2dm$ 3-tuples $(r_{ij}, v_{ij} + r_{ij}, -v_{ij} + r_{ij})_{i \in [d], j \in [2m]}$, with each r_{ij} chosen randomly from $\hat{k}$.
2. For each 3-tuple $(r_{ij}, v_{ij} + r_{ij}, -v_{ij} + r_{ij})$, Alice runs an OT_1^3 with Bob to get r_{ij} if $a_{ij} = 0$, $v_{ij} + r_{ij}$ if $a_{ij} = 1 \wedge a_i > 0$, and $-v_{ij} + r_{ij}$ otherwise.
3. Alice sends to Bob the sum of a_0 and those $2dm$ values she got. Bob subtracts $\sum_{i,j} r_{ij}$ from it to obtain $P(x_*)$.

Note that all the arithmetic are carried out in the system $\hat{n}$, which is large enough to guarantee that no error ever occurs. Then it's not hard to verify the correctness of this protocol, while its security is guaranteed by the following.

Lemma 9. *Protocol 4 is secure when parties are semi-honest.*

Proof. Alice's security proof is almost identical to that of Protocol 1, so we only discuss Bob's security here. Let $x_*, x'_* \in \hat{m}$, let E and E' be Bob's respective message distributions, and let E_k denote the distribution with the first k messages from E and the remaining messages from E'. Suppose E_{k_0-1} and E_{k_0} can be distinguished, for some k_0, and the 3-tuples in that k_0'th OT have the forms $(r, v+r, -v+r)$ and $(r', v'+r', -v'+r')$, for random r and r' and for some fixed v and v'. Let $\ell = (d+1)m$ and note that $v, v' \in \hat{\ell}$ because $2^{m-j}x^i \in \hat{\ell}$ for any $x \in \hat{m}$, $i \in [d]$ and $j \in [2m]$. Then according to Lemma 8, no matter which element Alice chooses, the two distributions of that element have a negligible distance. Using Lemma 1 and adapting Bob's security proof for Protocol 1, one can show that E and E' are indistinguishable. □

Note that the generalizations discussed in Section 3.4 also hold for floating-point numbers, and we have the following theorem.

Theorem 10. *Oblivious protocols exist for the problem of multi-variate polynomial evaluation (with random shares) over floating-point numbers.*

4.2 Neural Computing and Learning

There are several variants of the neural network model. We only demonstrate our result via 2-layer feedforward neural networks with back-propagation learning. Other variants can be handled similarly.

A 2-layer feedforward neural network has an internal layer of J nodes, with the j'th node having a weight vector $u_j = (u_{j1}, \ldots, u_{jI})$, and an output layer of K nodes, with the k'th node having a weight vector $w_k = (w_{k1}, \ldots, w_{kJ})$. Each node is associated with an activation function $f(z) = a \tanh(bz)$ (the hyperbolic tangent function). The network takes an input vector $x = (x_1, \ldots, x_I)$ and produces an output vector $o = (o_1, \ldots, o_K)$ in the following way.

Neural Computing

1. Compute $y_j = f(u_j \cdot x)$, for $j \in [J]$. Let $y = (y_1, \ldots, y_J)$.
2. Compute $o_k = f(w_k \cdot y)$, for $k \in [K]$.

The output vector o may not be correct, and a learning algorithm adjusts the weights according to how the vector o differs from the correct output vector d. The pair (x, d) constitutes a training example. The back-propagation learning (BP-Learning) algorithm adjusts the weights in the following way, with γ being some learning constant.

BP-Learning

For each training example in the training set, perform the following.

1. Compute $\delta_{ok} = \frac{b}{a}(d_k - o_k)(a^2 - o_k^2)$, for $k \in [K]$.
2. Compute $\delta_{yj} = \frac{b}{a}(a^2 - y_j^2)\sum_{k=1}^{K} \delta_{ok} w_{kj}$, for $j \in [J]$.
3. Update $w_{kj} = w_{kj} + \gamma \delta_{ok} y_j$, for $k \in [K], j \in [J]$.
4. Update $u_{ji} = u_{ji} + \gamma \delta_{yj} x_i$, for $i \in [I], j \in [J]$.

4.3 Oblivious Neural Computing and Learning

Now we want to carry out neural computing and neural learning in an oblivious way between two parties, Alice and Bob. Oblivious neural computing can be defined in a way similar to oblivious polynomial evaluation, except with Alice's polynomial replaced by a neural network. For oblivious neural learning, Bob has a set of training examples and wants to train Alice's neural network so that Bob knows nothing about Alice's neural network while Alice knows only what is implied by the weight changes. We need to be careful about Bob's security, as Alice's neural network has $IJ + JK$ weights and that many weight changes may reveal a lot to Alice. So we do not let Alice know the weight changes induced by each training example, and only let her get the overall weight changes after the training of all examples. Now a learning protocol is secure for Bob if Alice cannot distinguish two training sets that give the same overall weight changes. Note in practice, neural learning typically involves large training sets.

Another scenario is for Bob to keep random shares of those final weights, as long as he is willing to help Alice serve requests from other parties for oblivious neural computing. Later when another party wants to continue the training of Alice's neural network, Bob only needs to help with his shares for the first training example, and his duty is off after that. Contrary to the previous scenario, Alice cannot learn anything about Bob's training set in this way.

4.4 Oblivious Activation Function Evaluation

Here we discuss options for evaluating the activation function $f(z) = a\tanh(bz) = a(1 - \frac{2}{1+e^{2bz}})$ in an oblivious way. We will rely on an protocol for oblivious circuit evaluation [17, 9, 16], denoted as OCE, which is efficient for small circuits. Assume that Alice has x while Bob has y, and they want to generate random shares of $f(x + y)$ for Alice and Bob. One way is to use an OCE directly, if one can accept that the circuit for f is reasonably small. For cases allowing a large b, $f(z)$ is close to the threshold function, which has a very simple circuit, and again we can use OCE directly. Otherwise, we will approximate f in a piece-wise way by low degree polynomials and then apply our OPE protocol for it, which is described in the following. As f is smooth, there are intervals $I_0 = (-\infty, \ell_0], I_1 = (\ell_0, \ell_1], \ldots, I_n = (\ell_{n-1}, \infty)$, and degree-$d$ polynomials $P_0, P_1, \ldots, P_n$ such that

$$f(z) \approx P_i(z) \text{ for } z \in I_i,$$

for some small n and d, which seem good enough for practical purposes.[3] Let I be the function such that $I(z) = i$ for $z \in I_i$, which has a rather simple circuit and thus an efficient OCE protocol. Let $P_{i,x}(y) = P_i(x + y)$. Here is the oblivious protocol for evaluating the activation function.

Protocol 5

1. Alice generate random r_1. Bob runs OCE with Alice to get $r_2 = I(x + y) - r_1$.
2. Alice generate random s_1 and prepares the polynomial

$$Q_x(a, y) = -s_1 + \sum_{i=0}^{n} \frac{\prod_{j \neq i}(a + r_1 - j)}{\prod_{j \neq i}(i - j)} P_{i,x}(y).$$

Bob runs OPE with Alice for $s_2 = Q_x(r_2, y)$.

Note that Alice has s_1 and Bob has s_2 with $s_1 + s_2 = P_i(x + y)$ for $x + y \in I_i$, so the protocol is correct. The security proof is again similar to previous ones.

[3]For example, the error can be bounded by 2×10^{-6} with $n = 9$, $d = 9$, $\ell_0 = -7$, $\ell_8 = 7$, $P_0 = -1$, and $P_9 = 1$.

4.5 Oblivious Neural Algorithms

First we need to determine the possible range of floating-point numbers that can ever occur during computation. Then we can determine an appropriate floating-point number system $\hat{k}$ for random numbers and a system $\hat{n}$ for error-free arithmetics. Here is the protocol for oblivious neural computing which uses the OCE and OPE protocols with random shares.

Protocol 6

1. For $j \in [J]$, Alice and Bob compute random shares s_{j1}, s_{j2} of the inner product $u_j \cdot x$, and then compute random shares y_{j1}, y_{j2} of $y_j = f(s_{j1} + s_{j2})$. Let $y = (y_1, \dots, y_J)$.
2. For $k \in [K]$, Alice and Bob compute random shares t_{k1}, t_{k2} of $w_k \cdot y$, and then compute random shares o_{k1}, o_{k2} of $o_k = f(t_{k1} + t_{k2})$.

At the end, Alice can send her shares o_{k1} to Bob for him to obtain the output vector o. This is not needed for oblivious learning. Note that the protocol still works when the each weight vector is shared by two parties instead of owned by Alice, which is the case in oblivious learning.

Theorem 11. *Oblivious neural computing can be achieved by Protocol 6.*

Proof. The correctness is easy to verify. The security relies on the security of the protocol for oblivious polynomial evaluation with random shares and the protocol for oblivious evaluation of the activation function. Any breaking of Protocol 6's security gives a way for breaking one of the protocols which has been shown to be secure. □

An oblivious neural learning protocol can be derived similarly. Now only the protocol for OPE with random shares is needed.

Protocol 7

For each training example in the training set, perform the following.

1. Alice and Bob compute random shares of each $\delta_{ok} = \frac{b}{a}(d_k - o_k)(a^2 - o_k^2)$.
2. Alice and Bob compute random shares of each $\delta_{yj} = \frac{b}{a}(a^2 - y_j^2) \sum_{k=1}^{K} \delta_{ok} w_{kj}$.
3. Alice and Bob compute random shares of each $w_{kj} = w_{kj} + \gamma \delta_{ok} y_j$.
4. Alice and Bob compute random shares of each $u_{ji} = u_{ji} + \gamma \delta_{yj} x_i$.

After this, there are two alternatives, as described in Section 4.3. The first is to let Bob reveal his shares of those weights obtained in the last iteration, and let Alice derive the resulting neural network. The correctness is easy to verify. The security can be proved similarly as before. Now Alice cannot distinguish among training sets that give the same overall weight changes. The second alternative is not to have Bob give away his final shares to Alice, but for him to help Alice for her future task. In this way, Alice only obtains random shares of her new weights after each training example, including the final one. So each training example is secure and now Alice learns nothing about Bob's training set. So we have the following theorem.

Theorem 12. *Oblivious neural learning can be achieved by the combination of Protocol 6 and Protocol 7.*

Acknowledgements

We would like to thank Prof. Yuh-Dauh Lyuu for his help.

References

[1] M. Bellare and S. Micali, Non-interactive oblivious transfer and applications, in: Proc. CRYPTO '89, Lecture Notes in Computer Science, Vol. 435 (Springer, 1990), pp. 547–557.

[2] D. Boneh, Decision Diffie-Hellman problem, in: Proc. Algorithmic Number Theory 1998, Lecture Notes in Computer Science, Vol. 1423 (Springer, 1998), pp. 48–63.

[3] D. Bleichenbacher and P. Nguyen, Noisy polynomial interpolation and noisy chinese remaindering, in: Proc. EUROCRYPT 2000, Lecture Notes in Computer Science, Vol. 1807 (Springer, 2000), pp. 53–69.

[4] G. Brassard, D. Chaum, and C. Crepeau, Minimum disclosure proofs of knowledge, Journal of Computer and System Sciences 37(2), 1988, pp. 156–189.

[5] G. Brassard, C. Crepeau, and J. M. Robert, Information theoretical reductions among disclosure problems, in: Proc. 27th Ann. IEEE Symp. Foundations of Computer Science, 1986, pp. 168–173.

[6] D. Chaum, C. Crepeau, and I. Damgard, Multiparty unconditionally secure protocols (extended abstract), in: Proc. 20th Ann. ACM Symp. Theory of Computing, 1988, pp. 11–19.

[7] U. Feige, J. Kilian, and M. Naor, A minimal model for secure computation, in: Proc. 26th Ann. ACM Symp. Theory of Computing, 1994, pp. 554–563.

[8] Niv Gilboa, Two party RSA key generation, in: Proc. CRYPTO '99, Lecture Notes in Computer Science, Vol. 1666 (Springer, 1999), pp. 116–129.

[9] O. Goldreich, S. Micali, and A. Wigderson, How to play any mental game or a completeness theorem for protocols with honest majority, in: Proc. 19th Ann. ACM Symp. Theory of Computing, 1987, pp. 218–229.

[10] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby, Construction of a pseudo-random generator from any one-way function, SIAM Journal on Computing 28(4), 1999, pp. 1364–1396.

[11] R. Impagliazzo and D. Zuckerman, How to recycle random bits, in: Proc. 30th Ann. IEEE Symp. Foundations of Computer Science, 1989, pp. 248–253.

[12] Y. Ishai and E. Kushilevitz, Randomizing polynomials: a new representation with applications to round-efficient secure computaion, in: Proc. 41st Ann. IEEE Symp. Foundations of Computer Science, 2000, pp. 294–304.

[13] J. Kilian, Founding cryptography on oblivious transfer, in: Proc. 20th Ann. ACM Symp. Theory of Computing, 1988, pp. 20–31.

[14] Y. Lindell and B. Pinkas, Privacy preserving data mining, in: Proc. CRYPTO 2000, Lecture Notes in Computer Science, Vol. 1880 (Springer, 2000), pp. 36–54.

[15] M. Naor and B. Pinkas, Oblivious transfer and polynomial evaluation, in: Proc. 31st Ann. ACM Symp. Theory of Computing, 1999, pp. 245–254.

[16] T. Sander, A. Young, and M. Yung, Non-interactive cryptocomputing for NC^1, in: Proc. 40th Ann. IEEE Symp. Foundations of Computer Science, 1999, pp. 554–567.

[17] A. C. Yao, How to generate and exchange secrets, in: Proc. 27th Ann. IEEE Symp. Foundations of Computer Science, 1986, pp. 162–167.

[18] J. M. Zurada, Introduction to Artificial Neural Systems, PWS Publishing, 1994.

A Proof of Lemma 2

Let $\ell = |H||S_2|$. From Cauchy-Schwartz, $\sum_{h,v} |\Pr_{g,u}[(g, g(u)) = (h, v)] - 1/\ell|$ is at most

$$
\begin{aligned}
& \sqrt{\ell}\sqrt{\sum_{h,v}\left(\Pr_{g,u}[(g,g(u)) = (h,v)] - 1/\ell\right)^2} \\
= \quad & \sqrt{\ell\sum_{h,v}\Pr_{g,u}[(g,g(u)) = (h,v)]^2 - 2 + 1} \\
= \quad & \sqrt{\ell \Pr_{h,h',u,u'}[(h,h(u)) = (h', h'(u'))] - 1} \\
= \quad & \sqrt{\ell \Pr_{h,h'}[h = h'] \Pr_{h,u,u'}[h(u) = h(u')] - 1} \\
\leq \quad & \sqrt{\ell|H|^{-1}(\Pr_{u,u'}[u = u'] + \Pr_{h,u,u'}[h(u) = h(u')|u \neq u']) - 1} \\
= \quad & \sqrt{|S_2|\left(1/|S_1| + 1/|S_2|\right) - 1} \\
= \quad & \sqrt{|S_2|/|S_1|}.
\end{aligned}
$$