Fast Secure Logistic Regression for High Dimensional Gene Data

Martine De Cock[1]*, **Rafael Dowsley**[2], **Anderson Nascimento**[1]
Davis Railsback[1], **Jianwei Shen**[1], **Ariel Todoki**[1]
[1]School of Engineering and Technology
University of Washington Tacoma, Tacoma, WA 98402
{mdecock,andclay,drail,sjwjames,atodoki}@uw.edu
[2]Department of Computer Science
Bar-Ilan University, 5290002, Ramat-Gan, Israel
rafael@dowsley.net

Abstract

When collaboratively training Machine Learning (ML) models with Secure Multiparty Computation (SMC), the price paid for keeping the data of the parties private, is an increase in computational cost and runtime. A careful choice of ML techniques, algorithmic and implementation optimizations are a necessity to enable practical secure ML over distributed datasets. Such optimizations can be tailored to the kind of data and ML problem at hand. To the best of our knowledge, we present the fastest existing SMC implementation for training logistic regression models on high dimensional data. For our largest dataset, we train a model that requires over 7 billion secure multiplications; the training completes in about 2 hours in a local area network.

1 Introduction

Machine learning (ML) has many applications in the biomedical domain, such as diagnosis and personalized medicine. Biomedical datasets are typically characterized by high dimensionality, i.e. a high number of features such as lab test results or gene expression values, and low sample size, i.e. a small number of training examples corresponding to e.g. patients or tissue samples. Adding to these challenges, valuable training data is often split between parties (*data owners*) who cannot openly share the data because of privacy regulations and concerns.

We tackle the problem of training a binary classifier on high dimensional gene expression data held by different data owners, while keeping the training data private. This work is directly inspired by the iDASH2019 competition[2], which invited participants to design Secure Multiparty Computation (SMC) [4] solutions for collaborative training of ML models by two or more data owners. One of the competition datasets consists of 470 training examples (records) with 17,814 numeric features, while the other consists of 225 training examples with 12,634 numeric features. An initial 5-fold cross-validation analysis in the clear, i.e. without any encryption, indicated that in both cases logistic regression (LR) models are capable of yielding the level of prediction accuracy expected in the competition, prompting us to investigate SMC based protocols for secure LR training.

The iDASH2019 competition requirements imply the existence of multiple data owners who each send their training example(s) in an encrypted or secret shared form to *data processors* (computing nodes), as illustrated in Fig. 1. The *honest-but-curious* data processors are not to learn anything

*Guest Professor at Ghent University

[2]http://www.humangenomeprivacy.org/2019/competition-tasks.html, acc. on Aug 15, 2019

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

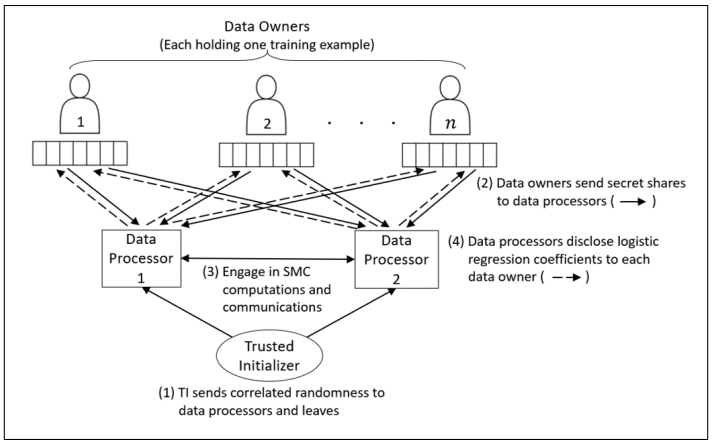


Figure 1: Overview of SMC based secure LR training. Each of n data owners secret shares their own training data between two data processors. The data processors engage in computations and communications to train a ML model, which is at the end revealed to the data owners.

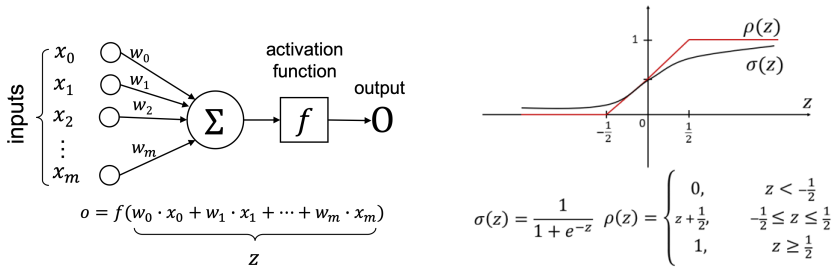
about the data as they engage in computations and communications with each other. At the end, they disclose the trained classifier – in our case, the coefficients of the LR model – to the data owners.

A variety of efforts have previously been made to train LR classifiers in a privacy preserving (PP) way. They can be grouped into outsourced computation approaches and multiparty computation approaches. In the outsourced setting, one party (the data owner) holds the data, while another party (the data processor), such as a cloud service, is responsible for the model training. Outsourced computation solutions usually rely on homomorphic encryption, with the data owner encrypting and sending their data to the data processor who performs computations on the encrypted data without having to decrypt it [2, 3, 8]. The multiparty computation setting that we consider differs from the above because we assume that the data resides with multiple data owners. Existing multiparty computation approaches to secure LR differ in the numerical optimization algorithms used for LR training and in the cryptographic primitives leveraged [7, 9, 10, 13]. Out of these, the closest to our work is the SecureML method [9]. The main novelty points of the SecureML paper are a clipped ReLu activation function (see Fig. 2(b)), a novel truncation protocol, and a combination of garbled circuits and secret sharing based computations in order to obtain a good trade-off between communication, computation and round complexities. The SecureML protocol is evaluated on a dataset with up to 5,000 features [9], while – to the best of our knowledge – the existing runtime evaluation of all other approaches for SMC based LR training is limited to 400 features or less [7, 13].

The main novelty points of our protocol for private LR training over a distributed dataset are: (i) a new protocol for securely computing the activation function that needs no secure comparison protocols; (ii) a novel way of bundling bit decomposition instantiations; and (iii) a very efficient implementation using state-of-the-art cryptographic engineering techniques in the programming language Rust – which turned out to be an excellent choice. To the best of our knowledge, ours is the very first implementation of SMC in Rust. When run in a local area network (LAN), SecureML reports 4.5ms for an activation evaluation. Our implementation can do 1024 activation evaluations in around 24ms. In summary, we designed a concrete solution for fast secure training of a binary classifier over gene expression data that meets the strict security requirements of the iDASH2019 competition. For our largest dataset, we train a model that requires over 7 billion secure multiplications; the training completes in about 2 hours in a LAN.

2 Method

In our secure LR training setting (cfr. Fig. 1), each data owner has a training example $d = (\mathbf{x}_d, t_d)$ in which $\mathbf{x}_d = \langle x_{d,1}, \ldots, x_{d,m} \rangle$ is an m-dimensional vector of real numbers, namely the values of m input attributes for example d, and $t_d \in \{0, 1\}$ is the ground truth class label. The data processors train a neuron to map the $\mathbf{x}_d$'s to the corresponding t_d's. As illustrated in Fig. 2(a), the neuron applies an activation function to a weighted sum of the inputs, to arrive at the output



(a) (b)

Figure 2: (a) Neuron; (b) Approximation of sigmoid activation function σ by clipped ReLu ρ

(1) **ALGORITHM** GRADIENTDESCENT(D, η)
(2) // Input: A set D with training examples $(\mathbf{x}_d, t_d)$; a learning rate η
(3) // Output: Weights w_i that minimize the sum of squared errors over the training data
(4) **for** $i \leftarrow 0$ **to** m **do**
(5) $\quad w_i \leftarrow 0$
(6) **until** termination condition is met **do**
(7) $\quad$ **for** $i \leftarrow 0$ **to** m **do**
(8) $\quad\quad \Delta w_i \leftarrow 0$
(9) $\quad$ **for each** $(\mathbf{x}_d, t_d)$ in D **do**
(10) $\quad\quad o_d \leftarrow \rho(w_0 \cdot x_{d,0} + w_1 \cdot x_{d,1} + \ldots + w_m \cdot x_{d,m})$
(11) $\quad\quad$ **for** $i \leftarrow 0$ **to** m **do**
(12) $\quad\quad\quad \Delta w_i \leftarrow \Delta w_i + \eta(t_d - o_d)x_{d,i}$
(13) $\quad$ **for** $i \leftarrow 0$ **to** m **do**
(14) $\quad\quad w_i \leftarrow w_i + \Delta w_i$

Figure 3: Full gradient descent algorithm for training a neuron with activation function ρ

$o_d = f(w_0 \cdot x_{d,0} + w_1 \cdot x_{d,1} + \ldots + w_n \cdot x_{d,m})$, in which $x_{d,0}$ has value 1 for all d's. The sigmoid activation function σ (cfr. Fig. 2(b)) that is traditionally used for LR, requires division and evaluation of an exponential function, which are expensive operations to perform in secret sharing based SMC, hence we approximate it with function ρ (cfr. Fig. 2(b)) as in [9]. To learn values for the weights from the training examples, the data processors follow the gradient descent based algorithm in Fig. 3.

Operations are performed on additive shares in a ring $\mathbb{Z}_q$ ($q = 2^\lambda$). The data owners first convert each feature value $x \in \mathbb{R}$ into a value $Q(x)$ in $\mathbb{Z}_q$ corresponding to a fixed point approximation of x with a fractional precision of a bits. We use a two's complement representation for negative numbers:[3]

$$Q(x) = \begin{cases} 2^\lambda - \lfloor 2^a \cdot |x| \rfloor & \text{if } x < 0 \\ \lfloor 2^a \cdot x \rfloor & \text{if } x \ge 0 \end{cases} \tag{1}$$

After the conversion, each data owner secret shares every number in its training example between the data processors (parties), henceforth called Alice and Bob. In general, a number z in $\mathbb{Z}_q$ is split in secret shares by picking $z_A, z_B \in \mathbb{Z}_q$ uniformly at random subject to the constraint that $z = z_A + z_B \mod q$, and then revealing z_A to Alice and z_B to Bob. We denote this secret sharing by $[\![z]\!]_q$, which can be thought of as a shorthand for (z_A, z_B). [5] has a detailed description of our notation.

Once they have received secret sharings $([\![\mathbf{x}_d]\!]_q, [\![t]\!]_q)$ of the training examples, the data processors perform a secure version of the training algorithm from Fig. 3. During the execution, they maintain a secret sharing $[\![\mathbf{w}]\!]_q$ of the weight vector learned so far, which is initially populated with 0s (lines 4-5). The data processors know in advance the agreed on learning rate η and the fixed number of iterations to perform during training (see line (6) in Fig. 3). We do not use *early stopping* because in that case the number of iterations would depend on the values in the training data, hence leaking information. The only operations in Fig. 3 that cannot be performed fully locally by the data processors, i.e. on their own local shares, are the computation of the inner product $\mathbf{w} \cdot \mathbf{x}$ in line 10, followed by the application of the activation function ρ in line 10, and the multiplication of $t_d - o_d$ with $x_{d,i}$ in line

[3]Note that the bit decomposition of $Q(x)$ is a bit string of length λ in which the most significant bit represents the sign of x, and the lowest a bits represent the fractional component of x. The bits in between are used to store the integer component of x, hence λ should be chosen large enough to be able to represent the maximum value produced during the LR protocol.

Table 1: Accuracy and training runtime for LR models

	# features	# pos. samples	# neg. samples	5-fold CV acc	runtime
BC-TCGA	17,814	422	48	99.56%	134.03 min
GSE2034	12,634	142	83	65.91%	48.02 min

12. For efficient secure multiplication, we use a trusted initializer (TI) that pre-distributes correlated randomness to the parties participating in the protocol (see Step (1) in Fig. 1). This *multiplication triplets* technique was originally proposed by Beaver [1] and is regularly used to enable very efficient solutions in the context of PPML (see e.g. [5, 6, 9, 11]). The TI is not involved in any other part of the execution and does not learn any data from the parties. Details about the corresponding multiplication protocol π_{DM}, which we use in line 12, can be found in [5]. It can be straightforwardly extended to a protocol for efficient computation of the inner product of two vectors, as used in line 10.

Finally, for the secure evaluation of the ρ on line 10, we propose a new protocol π_ρ that evaluates ρ directly over additive shares. At the beginning of this protocol, the parties have a secret sharing $[\![z]\!]_q$; at the end of the protocol they have a secret sharing $[\![\rho(z)]\!]_q$. Protocol π_ρ relies on the insight that $|z| \geq 0.5$ if, and only if, at least one of the b integer bits, or the most significant bit of the a fractional bits in the bit decomposition of z, is 1. A detailed description of π_ρ is given in Appendix A. As a further optimization, we exploit the fact that many evaluations of ρ need to be performed at once per iteration in Fig. 3, namely one per training example. To execute this efficiently, we have designed a novel protocol batch-π_{decomp} to combine multiple instances of the bit decomposition protocol π_{decomp} from [5] (see Appendix B). Bundling up several evaluations of ρ results in a highly optimized protocol, particularly in the scenario of LANs. SecureML reports 4.5ms for an activation evaluation over LAN. Our implementation completes 1024 evaluations in around 25ms.

3 Results

We implemented the protocols from Section 2 in Rust[4] and experimentally evaluated them on the BC-TCGA and GSE2034 datasets of the iDASH2019 competition. We refer to the competition website for more details about these datasets. We trained LR models on both datasets with a learning rate $\eta = 0.001$. The accuracy of the resulting models, evaluated with 5-fold cross-validation is presented in Table 1, along with the average runtime (in min) for training those models. It is important to note that these are the same accuracies that are obtained when training the neuron from Fig. 2a in the clear, i.e. there is *no accuracy loss* in the secure version.

We used[5] integer precision $b = 15$, fractional precision $a = 12$ and ring size $\lambda = 64$. We ran the experiments on AWS c5.9xlarge machines with 36 vCPUs, 72.0 GiB Memory. Each of the parties ran on separate machines (connected with a Gigabit Ethernet network), which means that the results in Table 1 cover communication time in addition to computation time. The results show that our implementation allows to securely train models with state-of-the-art accuracy [12] on the respective datasets within about 2 hours.

4 Conclusion

In this paper, we have described a novel protocol for implementing secure training of logistic regression over distributed parties using Secure Multiparty Computation. While being inspired by SecureML, our protocol and implementation present several novel points including: (i) a novel protocol for computing the activation function that does not require secure comparisons; (ii) a novel way to bundle many instances of secure bit decomposition and secure computations of the activation function; (iii) the very first optimized implementation of SMC in the Rust programming language. With our implementation, training a LR classifier over a highly dimensional dataset that requires over 7 billion secure multiplications, completes in about 2 hours. Our solution is particularly efficient for LANs where we can perform 1024 secure computations of the activation function in about 24ms. Evaluating our solution in a WAN or over the internet would be a next step of our work.

[4] https://bitbucket.org/mdecock/idash2019-rust/

[5] See Appendix C for an explanation for these choices.

References

[1] Donald Beaver. Commodity-based cryptography. In *STOC*, volume 97, pages 446–455, 1997.

[2] Charlotte Bonte and Frederik Vercauteren. Privacy-preserving logistic regression training. *BMC Medical Genomics*, 11(4):86, 2018.

[3] Hao Chen, Ran Gilad-Bachrach, Kyoohyung Han, Zhicong Huang, Amir Jalali, Kim Laine, and Kristin Lauter. Logistic regression over encrypted data from fully homomorphic encryption. *BMC Medical Genomics*, 11(4):81, 2018.

[4] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[5] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson Nascimento, Wing-Sea Poon, and Stacey Truex. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. *IEEE Transactions on Dependable and Secure Computing*, 16(2):217–230, 2019.

[6] Martine De Cock, Rafael Dowsley, Anderson C. A. Nascimento, and Stacey C. Newman. Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data. In *8th ACM Workshop on Artificial Intelligence and Security (AISec)*, pages 3–14, 2015.

[7] Khaled El Emam, Saeed Samet, Luk Arbuckle, Robyn Tamblyn, Craig Earle, and Murat Kantarcioglu. A secure distributed logistic regression protocol for the detection of rare adverse drug events. *Journal of the American Medical Informatics Association*, 20(3):453–461, 2012.

[8] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC Medical Genomics*, 11(4):83, 2018.

[9] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.

[10] Yuval Nardi, Stephen E Fienberg, and Robert J Hall. Achieving both valid and secure logistic regression analysis on aggregated data from different private sources. *Journal of Privacy and Confidentiality*, 4(1), 2012.

[11] Devin Reich, Ariel Todoki, Rafael Dowsley, Martine De Cock, and Anderson Nascimento. Privacy-preserving classification of personal text messages with secure multi-party computation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[12] Haozhe Xie, Jie Li, Qiaosheng Zhang, and Yadong Wang. Comparison among dimensionality reduction techniques based on random projection for cancer classification. *Computational Biology and Chemistry*, 65:165–172, 2016.

[13] Wei Xie, Yang Wang, Steven M Boker, and Donald E Brown. Privlogit: Efficient privacy-preserving logistic regression by tailoring numerical optimizers. *arXiv preprint arXiv:1611.01170*, 2016.

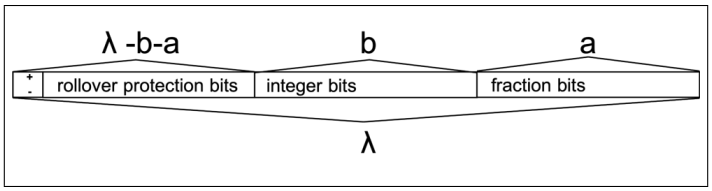


Figure 4: Register map of fixed-point representation of numbers shared over $\mathbb{Z}_{2^\lambda}$.

A Protocol π_ρ

As depicted in Figure 4, we use a fixed-point representation to share numbers over $\mathbb{Z}_{2^\lambda}$: a bits are allocated to represent the descending negative powers of 2 and b bits are allocated to represent increasing non-negative powers of 2. We use a bit-decomposition protocol π_{decomp} from [5] that converts additive sharings over a ring $\mathbb{Z}_{2^n}$ into n shares over the ring $\mathbb{Z}_2$. The idea of the protocol π_ρ that computes the activation function ρ on a value z is the following:

1. Using $[\![z]\!]_{2^\lambda}$ and the bit-decomposition protocol we obtain a sharing $[\![\mathsf{MSB}]\!]_{2^\lambda}$ corresponding to the most significant bit of z.
2. We obtain a secret sharing $[\![z_{rct}]\!]_{2^\lambda}$ corresponding to the absolute value of z by computing $[\![z_{rct}]\!]_{2^\lambda} \leftarrow (1 - [\![\mathsf{MSB}]\!]_{2^\lambda}) \cdot [\![z]\!]_{2^\lambda} + [\![\mathsf{MSB}]\!]_{2^\lambda} \cdot (2^\lambda - [\![z]\!]_{2^\lambda})$
3. $[\![z_{rct}]\!]_{2^\lambda}$ is bit-decomposed using $\pi_{decomp}(z_{rct})$ and the resulting secret sharings are separated into two relevant components:
   (a) All bits greater than or equal to $\frac{1}{2}$: $z_{int} = (z_{rct} >> (a-1)) \wedge (2^{b+1} - 1)$
   (b) All bits less than $\frac{1}{2}$, converted to additive shares: $z_{frac} = \pi^{-1}_{decomp}(z_{rct} \wedge (2^{a-1} - 1))$
4. The bitwise OR of z_{int} is logically equivalent to $|z| \geq \frac{1}{2}$. This determines whether $\rho(z)$ is in the constant or linear region. It can be computed efficiently as:
$$r_c = \neg \left(\bigwedge_{i=1}^{b+1} \overline{z_{int,i}} \right)$$
5. The MSB and r_c provide enough information to map z into the correct region of $\rho(z)$ as z_{frac} contains the relative shift between 0 and 1 if ρ is in the linear region. It can be computed as follows:
$$\rho(z) = \frac{1}{2} + \overline{\mathsf{MSB}} \cdot (\overline{r_c} \cdot z_{frac} + r_c \cdot \frac{1}{2}) + \mathsf{MSB} \cdot (2^\lambda - (\overline{r_c} \cdot z_{frac} + r_c \cdot \frac{1}{2}))$$

Algorithm 1: Evaluates a point on the curve of the activation function ρ over the ring $\mathbb{Z}_{2^\lambda}$.
Constraints: all values in $\mathbb{Z}_{2^\lambda}$ are representations of fixed point approximations of real numbers s.t. the lowest a bits represent the fractional component, the next b bits represent the integer component and $\lambda \geq 2(a+b)$. Further, a negative value x is represented as $2^\lambda - |x|$, where $|\cdot|$ is the absolute value of an integer.

SecureActivation $([\![z]\!]_{2^\lambda})$;
Input : $[\![z]\!]_{2^\lambda}$
Output : $[\![\rho(z)]\!]_{2^\lambda}$
Let $[\![msb]\!]_{2^\lambda} \leftarrow [\![z]\!]_{2^\lambda} >> \lambda - 1$
Let $[\![\frac{1}{2}]\!]_{2^\lambda} \leftarrow 1 << (a-1)$
$[\![z]\!]_{2^\lambda} \leftarrow (1 - [\![msb]\!]_{2^\lambda}) \cdot [\![z]\!]_{2^\lambda} + [\![msb]\!]_{2^\lambda} \cdot (2^\lambda - [\![z]\!]_{2^\lambda})$
Let $[\![d_{b+1}, ..., d_1]\!]_{2^\lambda} \leftarrow ([\![z]\!]_{2^\lambda} >> a-1) \wedge (2^{b+1} - 1)$ where $d_i \in \{0, 1\}$
Let $[\![L]\!]_{2^\lambda} \leftarrow [\![z]\!]_{2^\lambda} \wedge (2^{a-1} - 1)$
Let $[\![c]\!]_{2^\lambda} \leftarrow \bigvee_{i \in \{1, b+1\}} [\![d_i]\!]_{2^\lambda}$
Let $[\![r]\!]_{2^\lambda} \leftarrow [\![c]\!]_{2^\lambda} \cdot [\![\frac{1}{2}]\!]_{2^\lambda} + (1 - [\![c]\!]_{2^\lambda}) \cdot [\![L]\!]_{2^\lambda}$
Let $[\![r']\!]_{2^\lambda} \leftarrow (1 - 2 \cdot [\![msb]\!]_{2^\lambda}) \cdot [\![r]\!]_{2^\lambda} + [\![\frac{1}{2}]\!]_{2^\lambda}$
return $[\![r']\!]_{2^\lambda}$

B Protocol batch-π_{decomp}

The protocol π_{decomp} was proposed in [5]. It takes in a secret-shared value $[\![x]\!]_{2^\lambda}$ and outputs the bit decomposition $\{[\![x_\lambda]\!]_2, [\![x_{\lambda-1}]\!]_2 ..., [\![x_1]\!]_2\}$ works like the ripple carry adder arithmetic circuit based on the insight that the difference between the sum of two additive shares and an "XOR-sharing" of that sum is the carry vector. The fact that the carry bit of each bitwise sum must be present for the computation of the next bit implies a communication cost that is linear with the bit length λ. Obviously, this is an unacceptable cost to pay for every activation function. We propose batch-π_{decomp} as a means to take advantage of cases where many values need to be decomposed at once, as is the case in our LR protocol. It works by taking vertical slices of a collection of additive-shared values and transposing them into new integers. In this way, the carry of all 0-th order bits is computed, followed by all 1-st order bit carries, and so on up to the $(\lambda - 1)$-th carry. In this scheme, every group of λ ring elements requires only two sets of Beaver triples to compute the carries for the i-th bit. So until the optimal transmission buffer size is reached (based on our testing, 1024 bytes), the communication cost of performing multiple (4 times the transmission buffer size) bit decompositions is identical to only performing one. Additionally, the local computations are minimal and contribute virtually nothing to the running time.

C Truncation Errors

In our solution, we use the two-party offline truncation protocol for fixed point representations of real numbers proposed in [9]. This protocol always incurs an error of at most a bit flip in the least-significant bit. However, with probability $2^{a+1-\lambda}$, where a is the number of fractional bits, the resulting value is completely random. When this truncation protocol is performed on increasingly large data sets (in our case we run over 7 billion secure multiplications), the probability of an erroneous truncation becomes a real issue - an issue not significant in previous implementations. There are two phases in which truncation is performed: (1) when computing the dot product of the current weights vector with a training example, and (2) when the weights are updated at the end of a round. If a truncation error occurs during (1), the resulting erroneous value will be pushed into a reasonable range by the activation function and incur only a minor error for that round. If the error occurs during (2), an element of the weights vector will be updated to a completely random ring element and recovery from this error will be impossible. To mitigate this in experiments, we make use of 10-12 bits of fractional precision with a ring size of 64 bits, making the probability of failure $\frac{1}{2^{53}} < p < \frac{1}{2^{51}}$. The number of truncations that need to be performed was also reduced in our implementation by waiting to perform truncation until it is absolutely required. For instance, instead of truncating each result of multiplication between an attribute and its corresponding weight, a single truncation can be performed at the end of the entire dot product. Additional error is incurred on the accuracy by the fixed point representation itself. Through cross-validation with an in the clear implementation, we determined that 10 bits of fractional precision provide enough accuracy to make the output accuracy indistinguishable.