Secure multiparty computations in floating-point arithmetic

Chuan Guo, Awni Hannun, Brian Knott, Laurens van der Maaten,
Mark Tygert, and Ruiyu Zhu

July 31, 2020

Abstract

Secure multiparty computations enable the distribution of so-called shares of sensitive data to multiple parties such that the multiple parties can effectively process the data while being unable to glean much information about the data (at least not without collusion among all parties to put back together all the shares). Thus, the parties may conspire to send all their processed results to a trusted third party (perhaps the data providers) at the conclusion of the computations, with only the trusted third party being able to view the final results. Secure multiparty computations for privacy-preserving machine-learning turn out to be possible using solely standard floating-point arithmetic, at least with a carefully controlled leakage of information less than the loss of accuracy due to roundoff, all backed by rigorous mathematical proofs of worst-case bounds on information loss and numerical stability in finite-precision arithmetic. Numerical examples illustrate the high performance attained on commodity off-the-shelf hardware for generalized linear models, including ordinary linear least-squares regression, binary and multinomial logistic regression, probit regression, and Poisson regression.

1 Introduction

Passwords and long account and credit-card numbers are the dominant security measures, not because they are the most secure, but because they are the most conveniently implemented. Some data demands the highest levels of security and privacy protections, while for other data processing efficiency and sheer convenience are paramount — some security is better than none (which tends to be the alternative). The present paper proposes privacy-preserving, secure multiparty computations performed solely in the IEEE standard double-precision arithmetic that dominates most platforms for numerical computations. The scheme amounts to lossy, leaky cryptography, with the loss of accuracy and leakage of information carefully controlled via mathematical analysis and rigorous proofs. Information loss balances against roundoff error, providing perfect privacy at a specified finite precision of computations.

Perfect privacy at a given precision is when the information leakage is less than the specified precision (precision being limited due to roundoff error). The present paper provides perfect privacy at a precision of about 10^{-5} in the IEEE standard double-precision arithmetic of [13]; observing the encrypted outputs leaks no more than a millionth of a bit per input real number, whereas roundoff alters the results by around one part in a hundred thousand. In a megapixel image, the encrypted image would leak at most a single bit — enough information to discern whether the original image is dim or bright, perhaps, but no more. Performing all computations in floating-point arithmetic facilitates implementations on existing hardware, including both commodity central-processing units (CPUs) and graphics-processing units (GPUs), whereas alternative methods based on integer modular arithmetic could require difficult specialized optimizations to attain performance on par with the scheme proposed in this paper (even then, schemes based on integer modular arithmetic would have to contend with tricky issues of discretization and precision in order to handle the real numbers required for machine learning and statistics).

The algorithms and analysis consider the traditional *honest-but-curious* model of threats: we assume that the multiple parties follow the agreed-upon protocols correctly but may try to glean information from data they observe; the secure multiparty computations prevent any of the parties from gleaning much information without all parties conspiring together to break the scheme.

Our analysis provides no guarantees about what information leaks when all parties collude to reveal encrypted results. If all parties conspire to collect together all their shares or send them to a collecting agency, then the unified collection will reveal the secrets of whatever results get collected. If the collected information results from training a machine-learned model, then revealing the trained model can compromise the confidentiality of the data used to train that model, unless the model is differentially private. Ensuring privacy even after revealing the results of secure multiparty computations is complementary to securing the intermediate computations. The present paper only guarantees the privacy of the intermediate computations, providing no guarantees about what leaks when all parties collude to reveal the final results of their secure multiparty computations.

The work of [5] has a similar goal, pursued with markedly different methods and alternative mathematics (see, for example, the proof of Proposition 1 in Subsection 3.2 of the extended version at `http://eprint.iacr.org/2017/1234.pdf`); that work goes beyond ours by considering Newton-Raphson/Fisher-scoring/iteratively-reweighted-least-squares for logistic regression in the case when the testing set for testing the accuracy of an optimization for machine learning is identical to the training set.

This paper has the following structure: Section 2 introduces secure multiparty computations in floating-point arithmetic, reviewing classical methods such as additive sharing and Beaver multiplication. Section 3 upper-bounds the amount of information that can leak, referring to Appendices A, B, and C for full, rigorous proofs. Section 4 reviews techniques for efficient, highly accurate polynomial approximations to many real functions of interest (notably those in Table 3). Section 5 validates an implementation on synthetic examples and illustrates its performance on real measured data, too; the examples apply various generalized linear models, including ordinary linear least-squares regression, binary and multinomial logistic regression, probit regression, and Poisson regression. Appendices D, E, and F very briefly review Chebyshev series, minibatched stochastic gradient descent, and generalized linear models (including link functions), respectively — readers may wish to refer to those appendices as concise refreshers.

Throughout, all numbers and random variables are real-valued, even when not stated explicitly.

2 Secure multiparty computations

Secure multiparty computations allow holders of sensitive data to securely distribute so-called shares of their data to multiple parties such that the multiple parties can process the data without revealing the data and can only reconstruct the data by colluding to put back together (that is, to sum) all the shares. We briefly review an arithmetic scheme in the present section. The arithmetic scheme supports addition and multiplication, as discussed in the present section, as well as functions that polynomials can approximate accurately, as discussed in Section 4 below. To be concrete and simplify the presentation, we focus first on secure two-party computations, in Subsection 2.1, then sketch an extension to several parties in Subsection 2.2.

line	source	party 1	party 2
1	distributed data	$U-V$	V
2	distributed data	$X-Y$	Y
3	read from disk	$P-Q$	Q
4	read from disk	$R-S$	S
5	read from disk	$PR-T$	T
6	line 1 − line 3	$(U-V)-(P-Q)$	$V-Q$
7	line 2 − line 4	$(X-Y)-(R-S)$	$Y-S$
8	all reduce line 6	$U-P$	$U-P$
9	all reduce line 7	$X-R$	$X-R$
10	line 5 + (line 8)(line 4) + (line 3)(line 9) + (line 8)(line 9)/2	$(PR-T)+(U-P)(R-S)+$ $(P-Q)(X-R)+$ $(U-P)(X-R)/2$	$T+(U-P)S+$ $Q(X-R)+$ $(U-P)(X-R)/2$

Table 1: Ledgers for two parties in a Beaver multiplication of U and X

2.1 Two-party computations

We can hide a matrix X by masking with a random matrix Y, so that one party holds $X-Y$ and the other party holds Y. Given other data, say U, and another random matrix V hiding U, so that one party holds $U-V$ and the other party holds V, the parties can then independently form the sums $(U-V)+(X-Y)$ and $V+Y$ required to reconstruct $U+X=(U-V)+(X-Y)+(V+Y)$ if all these matrices have the same dimensions. This is known as "additive sharing," as described, for example, by [3]. Additive sharing thus supports privacy-preserving addition of U and X.

As introduced by [2], used by [3], and reviewed in Table 1, privacy-preserving multiplication of U and X is also possible whenever U and X are matrices such that their product UX is well-defined. Summing across the two parties in line 10 of Table 1 would yield $PR+(U-P)R+P(X-R)+(U-P)(X-R)=UX$, as desired. Notice that $U-P$ and $X-R$ in lines 8 and 9 of Table 1 are still masked by the random matrices P and R whose values are unknown to the two parties. The all-reduce in lines 8 and 9 requires the parties to conspire and distribute to each other the results of summing their respective shares of lines 6 and 7, but does not require the parties to get shares of any secret distributed data — that was necessary only for the original data U and X being multiplied (and this "original" data can be the result of prior secure multiparty computations). Also, all values on lines 3–5 are independent of U and X, so can be precomputed and stored on disk. As with addition, multiplication via the Beaver scheme never requires securely distributing secret shares, so long as the input data and so-called Beaver triples (P,R,PR) from Table 1 have already been securely distributed into shares. The secure distribution of shares is completely separate from the processing of the resulting distributed data set. Communication among the parties during processing is solely via the all-reduce in lines 8 and 9.

Table 1's scheme also works with matrix multiplication replaced by convolution of sequences.

Table 1 simplifies to Table 2 for the case when U and X are scalars such that $U=X$. Summing across the two parties in line 6 of Table 2 yields $P^2+2P(X-P)+(X-P)^2=X^2$, as desired. Notice that this recovers X^2 from a sum involving several terms as large as P^2; if $|X|\le 1<3<\gamma$ and $|P|\le\gamma$ (as well as $|T|\le\gamma^2$, where T cancels when summing across the two parties in line 6), then we obtain X^2 to precision upper-bounded by $6\gamma^2\cdot\varepsilon$, where ε denotes the machine precision (ε

line	source	party 1	party 2
1	distributed data	$X - Y$	Y
2	read from disk	$P - Q$	Q
3	read from disk	$P^2 - T$	T
4	line 1 − line 2	$(X - Y) - (P - Q)$	$Y - Q$
5	all reduce line 4	$X - P$	$X - P$
6	line 3 + (line 2)(line 5) · 2 + (line 5)^2/2	$(P^2 - T) + (P - Q)(X - P) \cdot 2 +$ $(X - P)^2/2$	$T + Q(X - P) \cdot 2 +$ $(X - P)^2/2$

Table 2: Ledgers for two parties in a Beaver squaring of X

is approximately 2.2×10^{-16} in the IEEE standard double-precision arithmetic of [13]). Theorem 3 proves that the information leakage may be as large as $6/\gamma$ if P, Q, and Y are distributed uniformly over $[-\gamma, \gamma]$ and T is distributed uniformly over $[-\gamma^2, \gamma^2]$ (similarly, P, Q, R, S, V, and Y in Table 1 should be distributed uniformly over $[-\gamma, \gamma]$ while T should be distributed uniformly over $[-\gamma^2, \gamma^2]$). Balancing the roundoff bound with the information bound requires $6\gamma^2 \cdot \varepsilon = 6/\gamma$, so that $\gamma = 1/\sqrt[3]{\varepsilon}$ (so $\gamma \approx 10^5$ for IEEE standard double-precision arithmetic).

2.2 Several-party computations

Extending Subsection 2.1 beyond two parties is straightforward. The steps in the algorithms, summarized in the columns labeled "source" in Tables 1 and 2, stay as they were (the division by 2 in the last line of Table 1 and in the last line of Table 2 becomes division by the number of parties). Distributing additive shares of data across several parties works as follows: for each piece of data to be distributed (in machine learning, a "piece" may naturally be a sample or example from the collection of all samples or examples), we generate n independent and identically distributed random matrices Y_1, Y_2, …, Y_n, where n is the number of parties (the number of parties need not relate to the total number of pieces, samples, or examples of data being distributed). We randomly permute the parties and then distribute to them (in that random order) $X + Y_1 - Y_2$, $Y_2 - Y_3$, $Y_3 - Y_4$, …, $Y_{n-1} - Y_n$, $Y_n - Y_1$, where X is the piece of data being shared. We generate different independent random variables and random permutations for different pieces of data. The distribution of the difference between independent random matrices drawn from the same distribution is the same for each party, making this an especially simple generalization to the case of several parties. Distributing additive shares to several parties leaks somewhat more information than limiting to only two parties; the present paper focuses on the case of two parties for simplicity.

3 Information leakage

This section bounds the amount of information-theoretic entropy that can leak when adding noise for masking, drawing heavily on canonical concepts from information theory, as detailed, for example, by [7].

We denote by X the scalar random variable that we want to hide, and by Y an independent variate that we add to X to effect the hiding. To simplify the analysis, we assume that the distribution of Y arises from a probability density function. Then, revealing $X + Y$ leaks the following number of bits of information about X:

$$h(X) - h(X \mid X + Y) = I(X;\ X + Y) = h(X + Y) - h(X + Y \mid X) = h(X + Y) - h(Y). \qquad (1)$$

In the left-hand side of (1), h denotes the Shannon entropy measured in bits if the distribution of X is discrete, and the differential entropy measured in bits (rather than nats) if the distribution of X is continuous. In the right-hand sides of (1), h denotes the differential entropy measured in bits (not nats); I denotes the mutual information. Given a prior on X, Bayes' Rule yields the full posterior distribution for X given $X + Y$; the information gain (or loss or leakage) defined in (1) is a summary statistic characterizing the divergence of the posterior from the prior.

Recall that mutual information is the fundamental limit on how much information can be gleaned from observing the outputs of a noisy channel; in our setting, we purposefully add noise in order to reveal only the results of communications via a (very) noisy channel, purposefully obscuring with noise the signal containing data being kept confidential and secure.

The information leakage is at most β/γ bits if $|X| \le \beta < \gamma$ and Y is distributed uniformly over $[-\gamma, \gamma]$, as stated in the following theorem and proven in Appendix A:

Theorem 1. *Suppose that X and Y are independent scalar random variables and β and γ are positive real numbers such that $|X| \le \beta < \gamma$ and Y is distributed uniformly over $[-\gamma, \gamma]$. Then, the information leaked about X from observing $X + Y$ satisfies*

$$I(X; X + Y) \le \frac{\beta}{\gamma}, \qquad (2)$$

where I denotes the mutual information between X and $X + Y$, measured in bits (not nats); the mutual information satisfies (1), which expresses I as a change in entropy. The inequality in (2) is an equality when X is β times a Rademacher variate, that is, $X = \beta$ with probability $1/2$ and $X = -\beta$ with probability $1/2$.

The following theorem, proven in Appendix B, states that the information leakage from hiding data multiple times is at most the sum of the information leaking from each individual hiding.

Theorem 2. *Suppose that X, Y, and Z are independent scalar random variables. Then,*

$$I(X;\ X + Y,\ X + Z) \le I(X;\ X + Y) + I(X;\ X + Z), \qquad (3)$$

where I denotes the mutual information.

The procedures of Tables 1 and 2 can also leak information, but not much — consider Table 2: Party 2 observes nothing about the input data X other than $X - P$, and Theorem 1 bounds how much information that reveals about X. Party 1 observes $X - Y$, $P - Q$, $P^2 - T$, $(X - Y) - (P - Q)$, $X - P$, and $(P^2 - T) + 2(P - Q)(X - P) + (X - P)^2/2$. The following theorem, proven in Appendix C, bounds how much information about X these observations reveal.

Theorem 3. *Suppose that X, Y, P, Q, and T are independent scalar random variables and γ is a positive real number such that $|X| \le 1 < 3 < \gamma$, the random variable T is distributed uniformly over $[-\gamma^2, \gamma^2]$, and Y, P, and Q are distributed uniformly over $[-\gamma, \gamma]$. Then,*

$$I\Big(X;\ X-Y,\ P-Q,\ P^2-T,\ (X-Y)-(P-Q),\ X-P,\ (P^2-T)+2(P-Q)(X-P)+(X-P)^2/2\Big) \le \frac{5}{\gamma} + \frac{1}{\gamma^2}, \qquad (4)$$

where I denotes the mutual information measured in bits, and its arguments (aside from X) are the observations in Table 2 under the column for "party 1."

Theorem 3 admits a formulation in the more general setting where $|X| \leq \beta$, as in Theorem 1, but the more specific case $|X| \leq 1$ in Theorem 3 suffices for the discussion in the present paper. The present paper makes no attempt at generality, except when convenient for our applications. The proof of Theorem 3 relies on the more general formulation (involving β) of Theorem 1 given above.

The following theorem states the classical data-processing inequality:

Theorem 4. *Suppose that random matrices X and Z are conditionally independent given a random matrix Y. Then,*

$$I(X;Z) \leq I(X;Y) \tag{5}$$

and

$$I(X;Z) \leq I(Y;Z), \tag{6}$$

where I denotes the mutual information.

The following is a corollary of Theorem 4:

Corollary 5. *Suppose that X and Y are random matrices and f is a deterministic function. Then,*

$$I(X;f(Y)) \leq I(X;Y), \tag{7}$$

where I denotes the mutual information.

Combining (1) and (7) shows that even very complicated manipulations such as the iterations in Section 4 below cause no further information leakage, despite changing the added noise in some highly nonlinear fashion: (7) guarantees that no information leaks beyond the individual maskings obeying (2)–(4), so long as the manipulations are deterministic algorithms (or randomized algorithms with randomization independent of the data being masked).

4 Polynomial approximations

Polynomials can approximate many functions of interest in machine learning, allowing the accurate approximation of those functions using only additions and multiplications. Section 2 above discusses schemes that multiple parties can use to perform additions and multiplications securely. The present section describes polynomial approximations useful in tandem with the schemes of Section 2. Subsection 4.1 leverages the method of Newton and Raphson. Subsection 4.2 uses Chebyshev series. Subsection 4.3 utilizes Padé approximation, in the method of scaling and squaring. The method of Newton and Raphson tends to be the most efficient, while Chebyshev series apply to a much broader class of functions. The method of scaling and squaring is for exponentiation. Table 3 lists the functions that each subsection treats.

4.1 Newton iterations

Various iterations derived from the Newton method for finding zeros of functions allow the computation of functions such as $\operatorname{sgn}(x)$, $1/x$, and $1/\sqrt{x}$ using only additions and multiplications (not requiring any divisions or square roots); in this subsection, x denotes a real number.

According to [15], the Newton-Schulz iterations for computing $\operatorname{sgn}(x)$ are

$$y_{k+1} = y_k(3 - y_k^2)/2, \tag{8}$$

$f(x)$	Name	Subsection
$\operatorname{sgn}(x)$	sign or signum	4.1
$\lvert x\rvert = x\operatorname{sgn}(x)$	absolute value	4.1
$1/x$	reciprocal	4.1
$1/\sqrt{x}$	raise to $-1/2$ power	4.1
$x^{-1/8}$	raise to $-1/8$ power	4.1
$\operatorname{ReLU}(x) = \max(x, 0)$	rectified linear unit	4.1
$\tanh(x)$	hyperbolic tangent	4.2
$1/(1 + \exp(-x))$	logistic	4.2
$\int_{-\infty}^{x} \exp(-y^2/2)\,dy \;/\; \sqrt{2\pi}$	CDF of standard normal	4.2
$\exp(x)$	exponential	4.3

Table 3: Subsections providing polynomial approximations to various functions

with $y_0 = x/\gamma$, where $|x| \le \gamma$ (and the desired loss of accuracy relative to the machine precision is less than a factor of γ).

According to [15], the Schulz (or Newton) iterations for computing $1/x$ when $x > 0$ are

$$y_{k+1} = y_k(2 - xy_k), \tag{9}$$

with $y_0 = 1$. Rescaling x (and then adjusting the resulting reciprocal) is important to align with the domain of convergence and high accuracy illustrated in Figure 1; and similar observations pertain to the rest of the iterations of the present subsection. In Subsection 4.3 below, x can range from 1 to the number of terms in the softmax (so requires scaling by the reciprocal of the number of terms in the softmax).

According to [11], the Newton iterations for computing $1/\sqrt{x}$ when $x > 0$ are

$$y_{k+1} = y_k(3 - xy_k^2)/2, \tag{10}$$

with $y_0 = 1$; similarly, the Newton iterations for computing $x^{-1/8}$ when $x > 0$ are

$$y_{k+1} = y_k(9 - xy_k^8)/8, \tag{11}$$

with $y_0 = 1$.

Figures 1–4 illustrate the errors obtained from (8)–(11); note that the scale of the vertical axes in Figures 1–3 involve 1e–16. In the figures, the tilde denotes the approximation computed via the Newton iterations (8)–(11); for example, $\widetilde{1/x}$ approximates $1/x$. The abscissae for each plot consist of 20,000 points equispaced on a logarithmic scale.

A common operation in the deep learning of [16] and others is the rectified linear unit

$$\operatorname{ReLU}(x) = \max(x, 0) = \frac{x(1 + \operatorname{sgn}(x))}{2}, \tag{12}$$

easily obtained from (8).

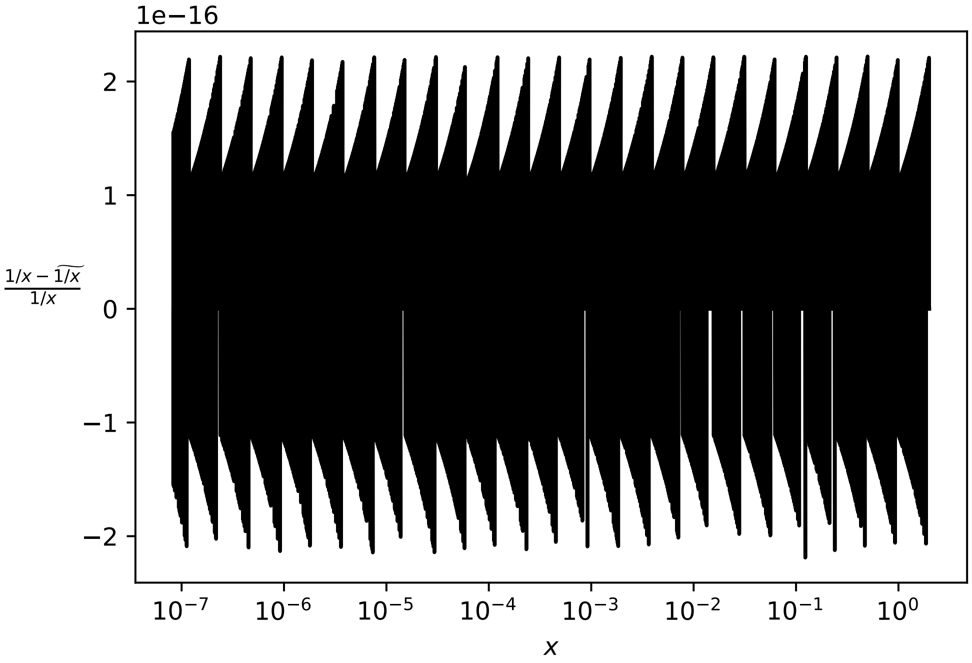

Figure 1: Relative error in computation of $1/x$ with 30 iterations of (9)

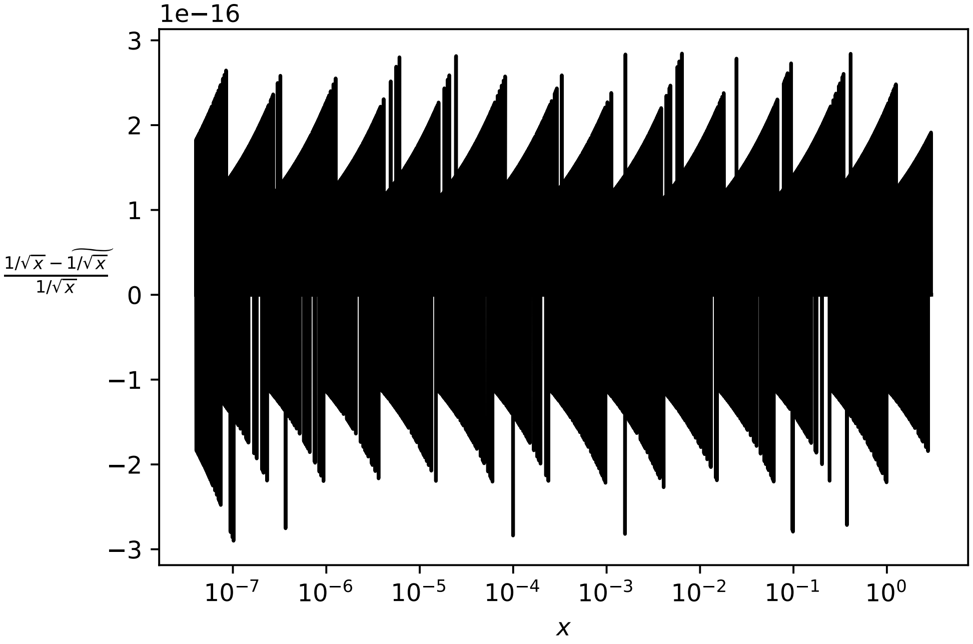

Figure 2: Relative error in computation of $1/\sqrt{x}$ with 26 iterations of (10)

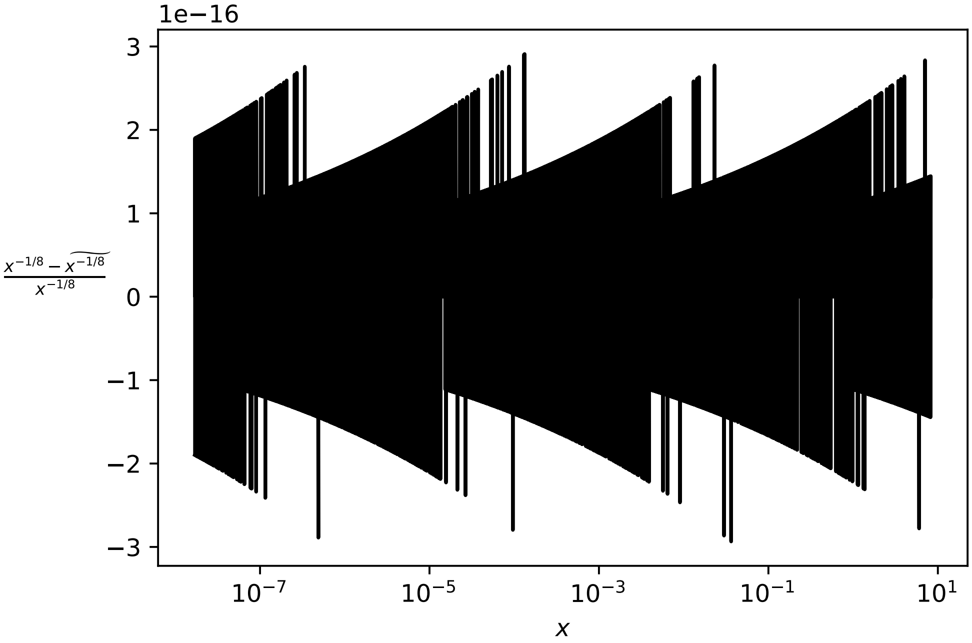

Figure 3: Relative error in computation of $x^{-1/8}$ with 24 iterations of (11)

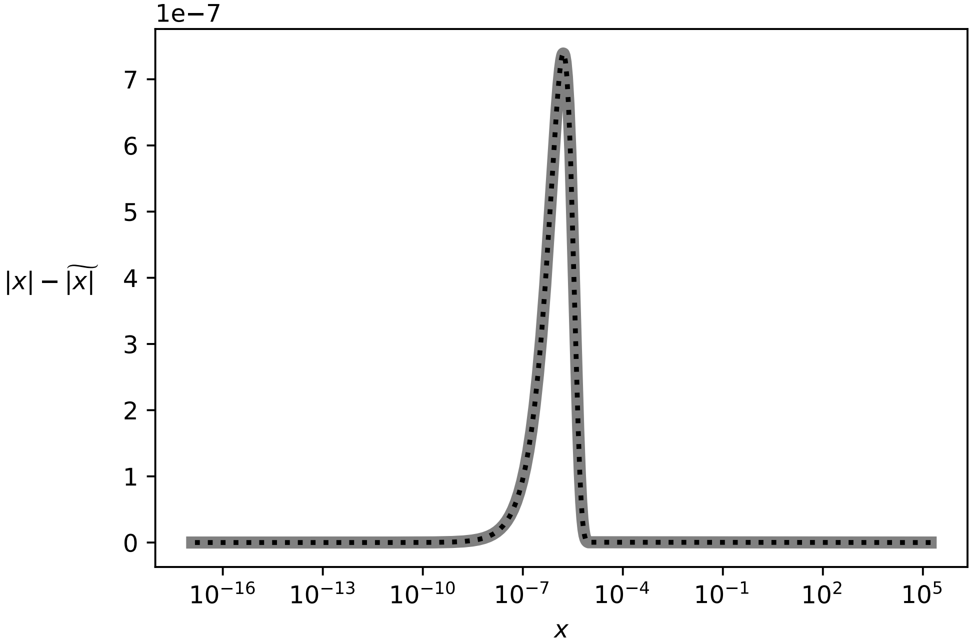

Figure 4: Absolute error in computation of $|x| = x\,\mathrm{sgn}(x)$ with 60 iterations of (8); the figure superimposes a black dotted line over a gray curve, where the black curve uses $y_0 = -x/\gamma$ to start (8) while the gray curve uses $y_0 = x/\gamma$, both with $\gamma = 10^5$

4.2 Chebyshev series

Chebyshev series provide efficient approximations to smooth functions using only additions and multiplications. The approximations are especially efficient for odd functions, such as

$$f(x) = \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}, \tag{13}$$

$$f(x) = \frac{1}{1 + \exp(-x)} - \frac{1}{2} = \tanh(x/2)/2, \tag{14}$$

and

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp(-y^2/2)\, dy - \frac{1}{2}; \tag{15}$$

in this subsection, x denotes a real number. The function in (13) is the hyperbolic tangent. The function in (14) is a constant plus the standard logistic function, familiar from logistic regression. The function in (15) is a constant plus the cumulative distribution function for the standard normal distribution, familiar from probit regression. Performing logistic regression or probit regression by maximizing the log-likelihood relies on the evaluation of (14) or (15), respectively, at least when using a gradient-based optimizer, the method of Newton and Raphson, or the method of scoring. Details about these functions and regressions are available, for example, in the monograph of [18].

Appendix D reviews algorithms for computing approximations via Chebyshev series of odd functions, with accuracy determined via two parameters, n and z, where the degree of the (odd) approximating polynomial is $2n - 1$, and $[-z, z]$ is the interval over which the approximation is valid. Setting $n = 50$ and $z = 10$ yields 7-digit accuracy for the approximation of (13); setting $n = 22$ and $z = 5$ yields 4-digit accuracy for the approximation of (14); and setting $n = 34$ and $z = 10$ yields 5-digit accuracy for the approximation of (15). In Section 5 below, we err on the side of caution, defaulting to $n = 60$ and $z = 20$ for (13) and (14) and to $n = 50$ and $z = 20$ for (15), while also discussing the results from other choices.

4.3 Softmax

As reviewed, for example, by [16], a common operation in machine learning is the so-called "softmax" transforming n non-positive real numbers x_1, x_2, ..., x_n into the n positive real numbers $\exp(x_1)/Z$, $\exp(x_2)/Z$, ..., $\exp(x_n)/Z$, where $Z = Z(x_1, x_2, \dots, x_n) = \sum_{k=1}^{n} \exp(x_k)$. These are the probabilities at unit temperature in the Gibbs distribution associated with energies $-x_1$, $-x_2$, ..., $-x_n$, where Z is the partition function. Once we have computed the exponentials, summation yields Z directly; the secure multiparty computations of Section 2 support such summation. Division by Z is available via the iterations in (9) of Subsection 4.1.

Thus, given real numbers x and β such that $-\beta \le x \le 0$, and a real number ε such that $0 < \varepsilon < 1$, we would like to calculate $\exp(x)$ to precision ε. We use the method of scaling and squaring, as reviewed, for example, by [12]. If we let n be the least integer that is at least $\log_2(2\beta^2/\varepsilon)$, then squaring $\exp(x/2^n)$ yields $\exp(x/2^{n-1})$, squaring $\exp(x/2^{n-1})$ yields $\exp(x/2^{n-2})$, and so on, so that n successive squarings will yield $\exp(x)$; further, $1+x/2^n$ approximates $\exp(x/2^n)$:

$$|\exp(x/2^n) - 1 - x/2^n| = \left| \sum_{k=2}^{\infty} (x/2^n)^k/k! \right| = (x/2^n)^2 \left| \sum_{k=0}^{\infty} (x/2^n)^k/(k+2)! \right| \le (x/2^n)^2 \exp(x/2^n), \tag{16}$$

that is,

$$1 + x/2^n = (1 + \delta)\exp(x/2^n), \quad |\delta| \le (x/2^n)^2, \tag{17}$$

while

$$\left|(1+\delta)^{2^n}-1\right| \le \left(1+(x/2^n)^2\right)^{2^n}-1 \le \left(\exp((x/2^n)^2)\right)^{2^n}-1 = \exp(x^2/2^n)-1 \le 2x^2/2^n \le \varepsilon, \quad (18)$$

so n successive squarings of $1+x/2^n$ yields $\exp(x)$ to relative accuracy ε (or better). The penultimate inequality in (18) follows from the fact that $\exp(y)-1 \le 2y$ for $0 \le y < 1/2$ (needless to say, $y = x^2/2^n \le \varepsilon/2 < 1/2$). We use $n = 20$ successive squarings in all numerical experiments of Section 5 below; this incurs an extra loss in relative accuracy of at most a factor of 2^{20} — about 6 digits — on account of roundoff error from adding 1 to $x/2^n$.

Given a real number $\gamma > 3\beta$, less than $6n\beta/\gamma$ bits can leak from computing the approximation to $\exp(x)$ if we add to $1 + x/2^n$ a random variable distributed uniformly over $[-\gamma/2^n, \gamma/2^n]$, and double the width of the added noise upon each of the n squarings, in accord with Theorems 1, 2, and 3 of Section 3.

Clearly, we can enforce that a real number x be non-positive by applying $-\operatorname{ReLU}(-x)$ from (12). Computing the softmax of real numbers that may not necessarily be non-positive is also possible, even without risk of leaking any information beyond the case for non-positive numbers. Indeed, we can replace each real number x_j with $x_j - \sum_{k=1}^n \operatorname{ReLU}(x_k)$, without altering the probability distribution produced by the softmax; we perform such replacement in our implementation of multinomial logistic regression. When implementing a softmax for multinomial logistic regression, we include a negative offset in the bias for the input to the softmax. That is, we adjust the bias to be a constant amount less, constant over all classes in the classification. Subtracting such a positive constant C tends to make $x_1 - C$, $x_2 - C$, …, $x_n - C$ negative even before replacing each real number $x_j - C$ with $x_j - C - \sum_{k=1}^n \operatorname{ReLU}(x_k - C)$. Subtracting a constant C reduces the sum $\sum_{k=1}^n \operatorname{ReLU}(x_k - C)$; without subtracting the constant, the sum $\sum_{k=1}^n \operatorname{ReLU}(x_k)$ can adversely impact accuracy if the sum becomes too large. Subtracting the constant C reduces accuracy by a factor of up to $\exp(C)$; we set $C = 5$ (so $\exp(C) \approx 148$ — a tad more than two digits) for our numerical experiments reported in the following section, Section 5.

5 Numerical examples

Via several numerical experiments, this section illustrates the performance of the scheme proposed above. All examples reported in the present section use two parties for the private computations. Subsection 2.2 outlines an extension to several parties. The terminology "in private" refers to computations fully encrypted via the scheme introduced above, while "in public" refers to computations in plaintext (meaning unencrypted — not protected or secured, but instead processed in the clear). Subsection 5.1 validates the scheme on examples for which the correct answer is known by construction. Subsection 5.2 applies the scheme to classical data sets.

All examples use minibatched stochastic gradient descent to maximize the log-likelihood under the corresponding generalized linear model, at the constant learning rates specified below (except where noted for probit regression). Appendix E briefly reviews stochastic gradient descent with minibatches and weight decay; Appendix F briefly reviews generalized linear models.

In all cases, we learn (that is, fit) not only the vector of weights to which the design matrix gets applied, but also a constant offset known as the "bias" in the literature on stochastic gradient descent. Thus, the linear function of the weight (fitted parameter) vector w in the generalized linear model is actually the affine transform $Aw + c$, where A is the design matrix and c is the bias vector whose entries are all the same constant offset, learned or fitted together with w during the iterations of stochastic gradient descent (whereas A stays fixed during the iterations). In multinomial logistic regression, the entries of the bias vector c are constant for each class (constant over all covariates and data samples), but the constant may be different for different classes.

In the subsections below, we view linear least-squares regression as a generalized linear model with the link function being the identity (please see Appendix F for a review of generalized linear models and link functions); equivalently, we take the parametric family defining the statistical model to be an affine transform of the vector of weights (parameters) plus independent and identically distributed normal random variables. The log-likelihood of a such a model summed over all samples in the design matrix A is simply a constant minus $\|Aw+c-b\|_2^2/2$, where $\|\cdot\|_2$ denotes the Euclidean norm, w is the vector of weights (parameters), $Aw+c$ is the affine transform defined by the design matrix A and bias vector c, and b is the vector of targets. For simplicity, when we report the "negative log-likelihood" or "loss" in plots, averaged over the m samples in the design matrix A, we report $\|Aw+c-b\|_2^2/m$, ignoring the constant and factor of 2.

The implementation of encrypted computations builds on CrypTen of [10], which in turn builds on PyTorch. The implementation uses only IEEE standard double-precision arithmetic. All experiments ran on one of Facebook's computer clusters for research, which enables rapid communication between the multiple parties. In actual deployments, communications between the multiple parties are likely to have high latency, dramatically impacting the speed of the multiparty computations. The speed of such communications would vary significantly between different applications and arrangements, likely requiring separate analyses and characterizations of computational efficiency for different deployments. Yet, while timings are fairly unique to the particular computational environment, the accuracies we report below should be fully representative for most applications.

5.1 Validations on synthetic data

For the synthetic examples discussed in the following sub-subsections, we set $m = 64$ and $n = 8$ for the numbers of rows and columns in the design matrices being constructed. Figure 5 displays the discrepancy of the computed results from the ideal solution (the ideal is known a-priori to two-digit accuracy by construction in these synthetic examples) as a function of the maximum value γ of the random variable distributed uniformly over $[-\gamma, \gamma]$. In accordance with the analysis in Subsection 2.1 above, the figure reports that $\gamma = 10^5$ works well, yielding the roughly two-digit agreement that would be optimal for the synthetic data sets constructed in the following sub-subsections, so we set $\gamma = 10^5$ for the remainder of the paper. The logistic and probit regressions reported below both rely on the Chebyshev approximations reviewed in Subsection 4.2 above, with the approximations being valid over the interval $[-20, 20]$ (so $z = 20$ in the notation of Subsection 4.2). Figure 6 indicates that the approximations whose polynomials have degrees less than 40 suffice for optimal accuracy, while polynomials with degrees less than 12 produce reasonably accurate results (accurate enough for most applications in machine learning for prediction, in which only residuals or matching targets matters). For the remainder of the paper, we use 120 terms in the Chebyshev approximation of the logistic function (or tanh), and 100 terms in the Chebyshev approximation of the inverse probit (or the cumulative distribution function of a standard normal variate). About half these 120 or 100 terms vanish in the Chebyshev approximations, since our Chebyshev approximations are odd functions. For both Figures 5 and 6, we trained for 10,000 iterations with 8 samples in each iteration's minibatch, thus sweeping through a random permutation of the full synthetic data set 1,250 times (each sweep is known as an "epoch"). The following sub-subsections detail the construction of our synthetic data sets.

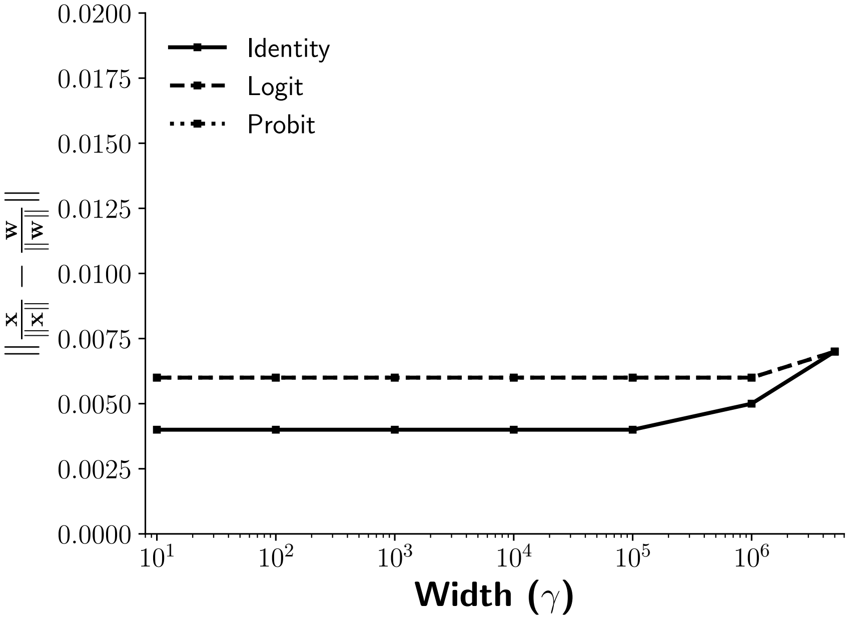

Figure 5: Euclidean norm of the difference between the ideal normalized weight vector $w/\|w\|_2$ and its computed approximation $x/\|x\|_2$ as a function of the width γ of the uniform noise on $[-\gamma, \gamma]$ added to the shares of data (the lines for the logit and probit links overlap)

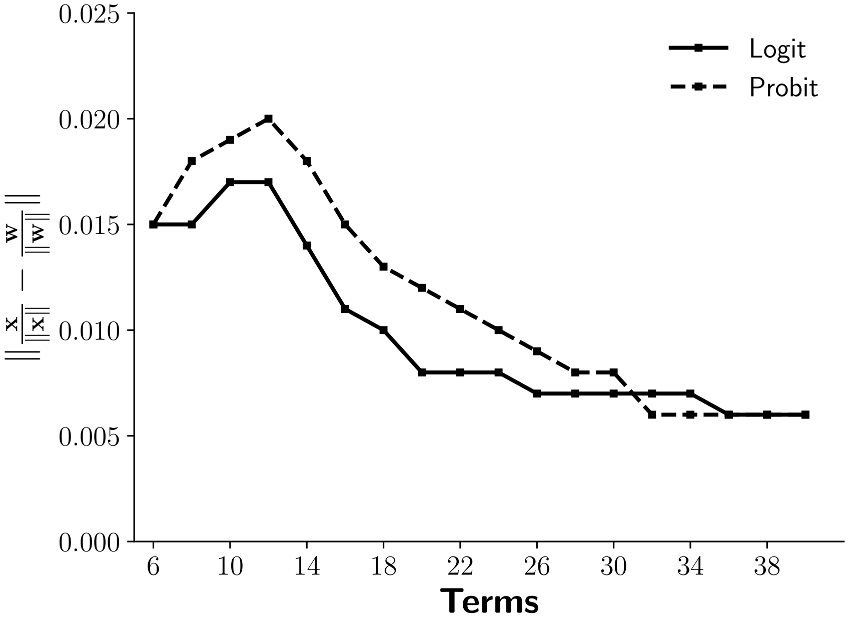

Figure 6: Euclidean norm of the difference between the ideal normalized weight vector $w/\|w\|_2$ and its computed approximation $x/\|x\|_2$ as a function of the number of terms in the Chebyshev approximation of Subsection 4.2 (about half the terms vanish, since the polynomial is odd)

5.1.1 Linear least-squares regression (identity link)

So that we can know a-priori the exact solution and minimal value (namely, 10) for the objective function being minimized — thus allowing us to check the accuracy of the computed solution — we form the design matrix A and target vector b as follows. Via Householder reflections, we orthonormalize the columns of an $m \times (n+1)$ matrix whose entries are all independent and identically distributed (i.i.d.) standard normal variates to obtain an $m \times n$ matrix A whose columns are orthonormal (A is the leftmost block of n columns) and an $m \times 1$ column vector v that is orthogonal to all columns of A and such that $\|v\|_2 = 1$ (v is the remaining column). We define the ideal weights w to be an $n \times 1$ column vector whose entries are i.i.d. standard normal variates. We define b to be the $m \times 1$ column vector

$$b = Aw + 10v \tag{19}$$

so that by construction

$$\min_x \|Ax - b\|_2^2 = \min_x \|Ax - Aw\|_2^2 + \|10v\|_2^2 = \|10v\|_2^2 = (10)^2. \tag{20}$$

Needless to say, obtaining $x = w$ drives $\|Ax - b\|_2$ to its minimum, 10.

We now consider some experiments with nontrivial settings of hyperparameters (such as a reasonably large number of iterations, minibatches with more than a single sample, and a learning rate that yields decent accuracy). After 10,000 iterations (which is 1,250 epochs) with 8 samples per minibatch at the learning rate 3×10^{-2}, the residuals $\|Ax + c - b\|_2$ obtained by training in public and by training in private are both equal to 10.0 to three significant figures. For training in private, the Euclidean norm of the difference between the ideal w and the computed weight vector x is 0.025 (the Euclidean norm of the difference between $w/\|w\|_2$ and $x/\|x\|_2$ is 0.004), and all entries of the vector c are -0.008 (which is -0.003 when divided by the Euclidean norm of x); that these values are so small certifies the correctness of the training. (After training in public, the Euclidean norm of the difference between the ideal w and the computed weight vector x is 0.027, and all entries of the vector c are 0.005, which are similar to those from training in private within the errors expected due to roundoff.)

5.1.2 Logistic regression (logit link)

Again, we set a-priori the exact solution to the minimization to be performed, letting us check the accuracy of the computed solution. Specifically, we define the ideal weights w to be the result of normalizing an $n \times 1$ column vector whose entries are i.i.d. standard normal variates, that is, we divide the vector whose entries are i.i.d. by its Euclidean norm, ensuring

$$\|w\|_2 = 1. \tag{21}$$

The construction of the design matrix A and target vector t is more involved; readers interested only in the results and not the details of the construction may wish to skip to the last two paragraphs of the present sub-subsection.

We now set up data for binary classification such that there are pairs of points straddling the ideal decision hyperplane separating classification into class 0 from classification into class 1, with each pair consisting of one point for class 0 and one point for class 1. We place the points in the pairs very close to each other, so that these pairs of points determine the decision hyperplane for classification to reasonably high accuracy. More precisely, for $j = 1, 2, \ldots, 10$, we construct an

$n \times 1$ column vector $v^{(j)}$ whose entries are i.i.d. standard normal variates, project off the component of $v^{(j)}$ along w to obtain $u^{(j)}$,

$$u^{(j)} = v^{(j)} - w \sum_{k=1}^{n} v_k^{(j)} w_k, \tag{22}$$

and set

$$A_{2j,k} = u_k^{(j)} + 0.02 w_k \tag{23}$$

and

$$A_{2j+1,k} = u_k^{(j)} - 0.02 w_k \tag{24}$$

for $k = 1, 2, \dots, n$. In addition to these pairs, we sprinkle in some other random points: for the remaining $m-20$ rows of A, we use $m-20$ rows from the result of orthonormalizing via Householder reflections the columns of an $m \times n$ matrix whose entries are all i.i.d. standard normal variates.

We construct the $m \times 1$ column vector

$$b = Aw \tag{25}$$

and define the target classes

$$t_j = \text{round}(\sigma(b_j)), \tag{26}$$

for $j = 1, 2, \dots, m$, where "round" rounds to the nearest integer (0 or 1) and σ is the standard logistic function

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \tag{27}$$

Combining (25), (23), (24), (22), and (21) yields

$$b_{2j} = \sum_{k=1}^{n} A_{2j,k} w_k = 0.02 \tag{28}$$

and

$$b_{2j+1} = \sum_{k=1}^{n} A_{2j+1,k} w_k = -0.02 \tag{29}$$

for $j = 1, 2, \dots, 10$. Since, for $j = 1, 2, \dots, 10$, b_{2j} is slightly positive while b_{2j+1} is slightly negative, the target classes t_{2j} and t_{2j+1} defined in (26) will be 1 and 0, respectively, even though the difference between the corresponding $(2j)$th and $(2j+1)$th rows of A is small — combining (23), (24), and (21) yields that the Euclidean norm of their difference is 0.04. The decision hyperplane separating class 0 from class 1 will thus have to pass between 10 pairs of points in n-dimensional space ($n = 8$), with the points in each pair very close to each other (albeit on opposite sides of the decision hyperplane). Classifying all these points correctly hence determines the hyperplane to reasonably high accuracy.

Needless to say, obtaining $x = w$ and $c = 0$ produces perfect accuracy for the logistic regression which classifies by rounding the result of (27) applied to each entry of $Ax+c$, as then $Ax = Aw = b$, and the target classes in t are the result of (27) applied to each entry of b. In fact, obtaining x as any positive multiple of w together with $c = 0$ yields perfect accuracy — any positive multiple of a vector orthogonal to the hyperplane separating the two classes specifies that same hyperplane.

We again consider some experiments with nontrivial settings of hyperparameters (such as a reasonably large number of iterations, minibatches with more than a single sample, and a learning rate that yields decent accuracy). After 10,000 iterations (which is 1,250 epochs) with 8 samples per minibatch at the learning rate 3, training binary logistic regression in private drives the Euclidean norm of the difference between $x/\|x\|_2$ and the ideal $w/\|w\|_2$ to 0.006 and drives every entry of $c/\|x\|_2$ to 0.006, where c is the vector of offsets (whose entries are all the same). That these values are so small validates the training. The log-likelihood, averaged over all $m = 64$ samples, changes from -0.785 to -0.088 over the 10,000 iterations (needless to say, the log-likelihood cannot exceed 0). The accuracy becomes exactly perfect (that is, 1). (Training in public drives the Euclidean norm of the difference between $x/\|x\|_2$ and the ideal $w/\|w\|_2$ to 0.007 and drives every entry of $c/\|x\|_2$ to 0.007, while the average log-likelihood becomes -0.086 and the accuracy becomes perfectly 1 over the 10,000 iterations, all results which are similar to the private results to within errors expected due to roundoff.)

When training multinomial logistic regression for two classes on the same synthetic data set with the same settings, similar validation attains: for training in private, the Euclidean norm of the difference between the ideal $w/\|w\|_2$ and the computed $x/\|x\|_2$ becomes 0.008, and every entry of $c/\|x\|_2$ becomes either -0.037 or -0.033, where c contains the bias offsets. The log-likelihood, averaged over all $m = 64$ samples, changes from -0.958 to -0.047 over the 10,000 iterations, and the accuracy becomes perfect (that is, exactly 1). Of course, using multinomial logistic regression with a binomial distribution rather than standard logistic regression makes no sense, but these results certify the correctness of the multinomial logistic regression nonetheless. (Training in public drives the Euclidean norm of the difference between the ideal $w/\|w\|_2$ and the computed $x/\|x\|_2$ to 0.006, drives every entry of $c/\|x\|_2$ to -0.039 or -0.033, drives the average log-likelihood to -0.046, and drives the accuracy to perfectly 1 over the 10,000 iterations, all results which match those from training in private to within the errors expected due to roundoff.)

5.1.3 Probit regression (probit link)

The design matrix A and target vector t for probit regression will be the same as for logistic regression, but replacing the sigmoid σ defined in (27) with the inverse probit

$$\sigma(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp(-y^2/2)\, dy, \tag{30}$$

which is also known as the cumulative distribution function for standard normal variates. An iteration of stochastic gradient descent involves selecting rows of the design matrix A and collecting them together into a matrix R, as well as collecting together the corresponding targets from t into a vector s (the number of rows is the size of the minibatch). One iteration of stochastic gradient descent for maximizing the log-likelihood in logistic regression updates the weight vector x by adding to x the learning rate η times the transpose $R^\top$ applied to the difference between the corresponding target samples s and σ defined in (27) applied to each entry of $Rx + c$; with a minor abuse of notation, we could write that x updates to $x + \eta R^\top(s - \sigma(Rx + c))$. Since the sigmoid defined in (30) is numerically very similar to the sigmoid defined in (27) (after scaling such that the variances of the sigmoids are the same), we use the same updating formula for probit regression as for logistic regression, but using the design matrix A associated with probit regression rather than that for logistic regression, and using the sigmoid σ associated with probit regression rather than that for logistic regression. A naïve application of stochastic gradient descent for maximizing the log-likelihood in probit regression would update the weight vector in the same direction, but scaled slightly, effectively altering the learning rate a tiny bit from iteration to iteration; we omit the extra computations required to follow the naïve method exactly.

As with logistic regression, obtaining $x = w$ and $c = 0$ produces perfect accuracy for the probit regression which classifies by rounding the result of (30) applied to each entry of $Ax+c$. And, again, obtaining x as any positive multiple of w together with $c = 0$ yields perfect accuracy — any positive multiple of a vector orthogonal to the hyperplane separating the two classes specifies that same hyperplane. Once again we consider some experiments with nontrivial settings of hyperparameters (such as a reasonably large number of iterations, minibatches with more than a single sample, and a learning rate that yields decent accuracy). After 10,000 iterations (which is 1,250 epochs) with 8 samples per minibatch at the learning rate 3, training in private drives the Euclidean norm of the difference between $x/\|x\|_2$ and the ideal $w/\|w\|_2$ to 0.006 and drives every entry of $c/\|x\|_2$ to 0.005, where c is the vector of offsets (whose entries are all the same). That these values are so small validates the training. The log-likelihood, averaged over all $m = 64$ samples, changes from -0.918 to -0.058 over the 10,000 iterations (needless to say, the log-likelihood cannot exceed 0). The accuracy becomes precisely perfect (that is, 1). (Training in public drives the Euclidean norm of the difference between $x/\|x\|_2$ and the ideal $w/\|w\|_2$ to 0.007 and drives every entry of $c/\|x\|_2$ to 0.006, while the average log-likelihood becomes -0.056 and the accuracy becomes perfectly 1 over the 10,000 iterations, all results which match the private results to within errors expected due to roundoff.)

5.1.4 Poisson regression (log link)

To construct a data set for Poisson regression in which we know a-priori the ideal solution to reasonably high accuracy (so that we can easily verify the accuracy of a computed solution), we spread over a substantial range of integers the target counts being fitted in the Poisson regression. So long as the target counts cover a sufficiently wide range of integers, with most counts not too small, the discrete spacing of the counts will cause little change from the solution to an undiscretized (continuous) regression. Thus, if we set up a continuous regression for which we know the exact solution a-priori, and round the targets (regressands) in that regression, then the solution to the associated Poisson regression should be the same to reasonably high accuracy. For this purpose, we construct the ideal solution vector w such that its Euclidean norm is substantial — 10 — and add an offset of 3 to every entry of the vector b whose entrywise exponentiation is the vector of target counts (after rounding). The 10 will ensure that the range of targets is sufficiently wide, and the 3 will ensure that most of the target counts are not too small.

More concretely, we obtain the design matrix A by orthonormalizing via Householder reflections the columns of an $m \times n$ matrix whose entries are i.i.d. standard normal variates. We define w to be 10 times the result of normalizing an $n \times 1$ column vector whose entries are i.i.d. standard normal variates, that is, we divide the vector whose entries are i.i.d. by its Euclidean norm, and then multiply by 10, ensuring

$$\|w\|_2 = 10. \tag{31}$$

We construct the $m \times 1$ column vector

$$b = Aw + 3, \tag{32}$$

where "3" indicates the $m \times 1$ column vector whose entries are all 3. We then define the target counts

$$t_j = \text{round}(\exp(b_j)), \tag{33}$$

for $j = 1, 2, \dots, m$, where "round" rounds to the nearest integer $(0, 1, 2, \dots)$. Using 10 as the norm of w in (31) ensures that the integers $t_1, t_2, \dots, t_m$ vary over a significant range, while using 3 in the right-hand side of (32) ensures that, on average, half will be greater than $\exp(3) \approx 20$. Having targets that vary over a significant range and with many not too small ensures that the maximum-likelihood estimates of w and 3 in Poisson regression are close to w and 3 with reasonably high accuracy — the discretization from the rounding operation in (33) matters little when many counts are large and spread over a range significantly greater than the discretization spacing.

As in the experiments reported above, we conduct further experiments at nontrivial settings of hyperparameters (such as a reasonably large number of iterations, minibatches with more than a single sample, and a learning rate that yields decent accuracy). After 10,000 iterations (which is 1,250 epochs) with 8 samples per minibatch at the learning rate 3×10^{-3}, training in private drives the Euclidean norm of the difference between $x/\|x\|_2$ and the ideal $w/\|w\|_2$ to 0.003 and drives every entry of $c - 3$ to 0.003, where c is the vector of offsets (whose entries are all the same). That these values are so small certifies the correctness of the training, as does the following result: the log-likelihood for the obtained weight vector x and offset c, averaged over all $m = 64$ samples, is -2.349 after the 10,000 iterations, which matches the log-likelihood for the ideal weight vector w and ideal offset (3) to three-digit precision. (Training in public for 10,000 iterations drives the average log-likelihood, every entry of $c - 3$, and the Euclidean norm of the difference between $x/\|x\|_2$ and the ideal $w/\|w\|_2$ to the same values to three-digit absolute precision as from training in private.)

5.2 Performance on measured data

The following sub-subsections illustrate the application of the scheme proposed above to several benchmarks, namely, handwritten digits from the data set known as "MNIST," forest covers from the data set known as "covtype," and the numbers of deaths from horsekicks in corps of the Prussian army over two decades.

5.2.1 MNIST

We use both binary and multinomial logistic regression, as well as probit regression, to analyze a classic data set of handwritten digits, created by Yann LeCun, Corinna Cortes, and Christopher J. C. Burges via merging two sets from the National Institute of Standards and Technology; the mixed NIST set is available at `http://yann.lecun.com/exdb/mnist` as a training set of 60,000 samples and a testing set of 10,000 examples. Each sample is a centered 28-pixel × 28-pixel grayscale image of one of the digits 0–9, together with a label for which one; pixel values can range from 0 to 1. For multinomial logistic regression we use all 10 classes (with one class per digit); for binary logistic and probit regressions, we use the 2 classes corresponding to the digits 0 and 1, for which there are 12665 samples in the training set, and 2115 samples in the testing set. To mimic common settings for processing MNIST, we set the number of samples in a minibatch to 50 for training with all 10 classes, and to 85 for training with only the 2 classes corresponding to the digits 0 and 1. To produce decent accuracy after training, we trained for 20 epochs (that is, 24,000 iterations) at learning rate 10^{-2} with all 10 classes, and for 30 epochs (that is, 4,470 iterations) at learning rate 10^{-3} with only the 2 classes corresponding to the digits 0 and 1. During training for all 10 classes, we supplemented each iteration of stochastic gradient descent with weight decay of 10^{-3}, which is equivalent to adding to the objective function being minimized (that is, to the negative log-likelihood) a regularization term of 10^{-3} times half the square of the Euclidean norm of the weights. This weight decay has negligible impact on accuracy yet ensures that the constant 5 we subtract from the bias in the stochastic gradient descent is sufficient to make almost all inputs of the softmax be non-positive, as suggested in the last paragraph of Subsection 4.3 above.

Figure 7 plots the results of training and Figure 8 displays the performance of the resulting trained model when applied to the testing set, both as a function of the maximum value γ of the random variable distributed uniformly over $[-\gamma, \gamma]$; the figures show that $\gamma = 10^5$ works well, in

link	train loss	test loss	train accuracy	test accuracy
identity	0.063	0.057		
logit	0.039	0.033	0.997	0.999
probit	0.025	0.019	0.997	0.999

Table 4: Values of the negative log-likelihood (the "loss") and accuracy averaged over the training samples or testing samples from MNIST, with $\gamma = 10^5$ being the maximal possible value of the random variable distributed uniformly over $[-\gamma, \gamma]$ added to shares of the data; the identity link does not solve a classification problem, so has no natural notion of accuracy for MNIST

agreement with the analysis in Subsection 2.1 above. Table 4 details the results for $\gamma = 10^5$; training in public produces the same results at the precision reported in the table. For this application to machine learning, even $\gamma = 10^6$ produces practically perfect predictions during both training and testing. Logistic regression corresponds to the logit link; probit regression corresponds to the probit link. The value of the log-likelihood averaged over the testing set is remarkably similar to the average value over the training set, showing that training generalizes well to the independent testing set. Figures 7 and 8 and Table 4 also include the results of using the identity link, just to check (of course the identity link results in real numbers that may not be limited to the values 0 and 1 corresponding to the digits 0 and 1 — the identity link does not perform a classification, even though classification is the natural task for this data set). The identity link runs fine, though the resulting negative log-likelihoods ("loss") have no obvious interpretation in this scenario.

Figure 9 displays the results of training multinomial logistic regression for all 10 classes (one class per digit), along with applying the resulting trained model to the testing set, as a function of the maximum γ of the random variable distributed uniformly over $[-\gamma, \gamma]$; the accuracy exceeds 0.9 from $\gamma = 10^3$ to $\gamma = 10^6$. The generalization from the training set to the testing set is ideal. At $\gamma = 10^5$, the training loss is 0.318, the testing loss is 0.305, the training accuracy is 0.913, and the testing accuracy is 0.917; training in public yields the same results to three-digit precision.

5.2.2 Forest cover type

We use both binary and multinomial logistic regression, as well as probit regression, to analyze standard data on the type of forest cover based on cartographic variates from Jock A. Blackard of the United States Forest Service, Denis J. Dean of the University of Texas at Dallas, and Charles W. Anderson of Colorado State University (with copyright retained by Jock A. Blackard and Colorado State University); the original data is available from the archive of [9] at `http://archive.ics.uci.edu/ml/datasets/covertype` and the preprocessed and formatted versions that we use are available from the work of [6] at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#covtype.binary` (for binary classification) and `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#covtype` (for the multinomial logistic regression).

We predict one of 7 types of forest (or one of 2 for the binarized data) based on 10 integer-valued covariates (elevation, aspect, slope, horizontal and vertical distances to bodies of water, horizontal distance to roadways, horizontal distance to fire points, and hillshade at 9am, 12pm, and 3pm), as well as one-hot encodings of 4 types of wilderness areas and 40 types of soil. Thus, there are 54

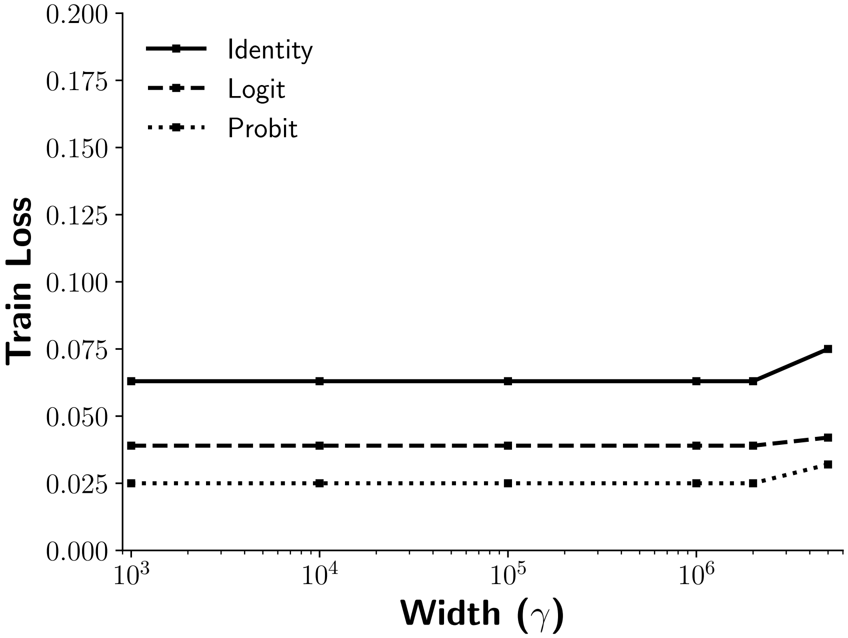

Figure 7: Value of the negative log-likelihood (the "loss") averaged over the training data from MNIST, after convergence of the training iterations, as a function of the maximum value γ of the random variable distributed uniformly on $[-\gamma, \gamma]$ added to the shares of the data

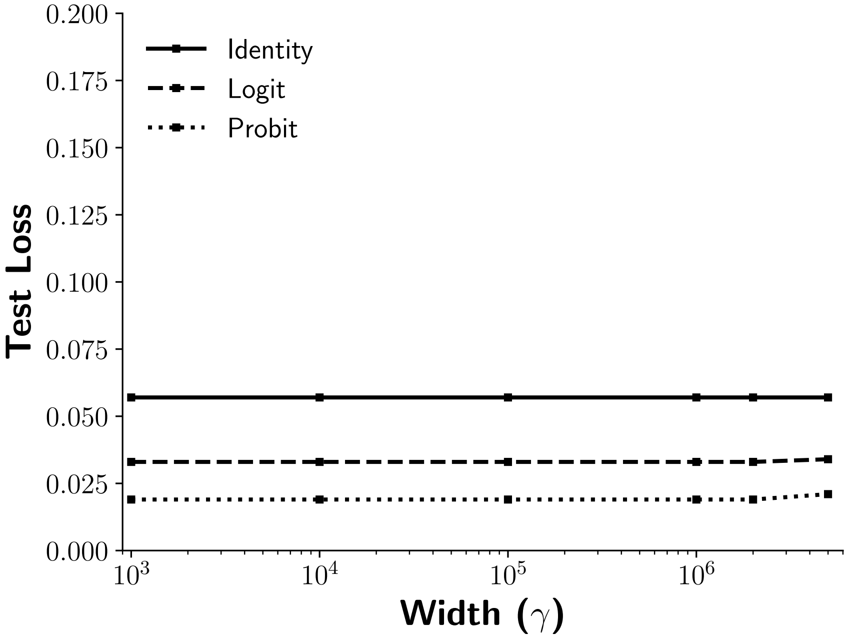

Figure 8: Value of the negative log-likelihood (the "loss") averaged over the testing data from MNIST, as a function of the maximum value γ of the random variable distributed uniformly on $[-\gamma, \gamma]$ added to the shares of the data

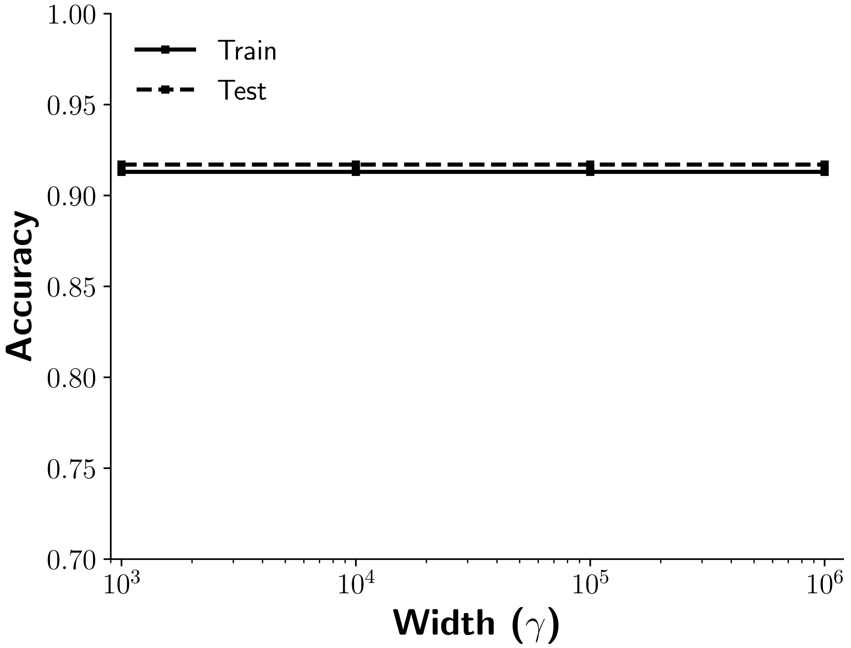

Figure 9: Accuracy of multinomial logistic regression averaged over the training or testing data from MNIST, as a function of the maximum value γ of the random variable distributed uniformly on $[-\gamma, \gamma]$ added to the shares of the data

covariates in all, including the one-hot encodings. (A one-hot encoding is a vector whose entries are all 0 except for one 1 in the position corresponding to the associated type.) For normalization, we subtract the mean from each of the integer-valued covariates (not from the one-hot encodings) and then divide by the maximum absolute value. We randomly permute and then partition the data into a training set of 500,000 samples and a testing set of the other 81,012 samples. To yield good accuracy after training, we trained for 20 epochs (that is, 10,000 iterations) with 1,000 samples per minibatch (the large minibatch was feasible with this particular data set). We adjusted the learning rate to ensure good accuracy after training, setting the learning rate for the binary classification to 3 (for identity link to 0.1) and for the multi-class (7-class) classification to 1. During training for all 7 classes, we supplemented each iteration of stochastic gradient descent with weight decay of 10^{-3}, to be consistent with our training for all 10 classes of MNIST in the previous sub-subsection. This weight decay barely impacts accuracy yet makes the constant 5 that we subtract from the bias in the stochastic gradient descent shift almost all inputs of the softmax to be non-positive, as suggested in the last paragraph of Subsection 4.3 above.

Figure 10 displays the results of training and Figure 11 depicts the performance of the resulting trained model when applied to the testing set, both as a function of the width γ of the random variable distributed uniformly over $[-\gamma, \gamma]$; the figures show that $\gamma = 10^5$ works well, in accordance with the analysis in Subsection 2.1 above. Table 5 details the results for $\gamma = 10^5$; training in public yields the same results to within ±0.001 of those reported in the table. In fact, even $\gamma = 10^6$ works perfectly fine for this application to machine learning. Logistic regression corresponds to the logit link; probit regression corresponds to the probit link. The value of the log-likelihood averaged over the testing set is reassuringly close to the average value over the training set, demonstrating good

link	train loss	test loss	train accuracy	test accuracy
identity	0.175	0.176		
logit	0.515	0.515	0.755	0.758
probit	0.517	0.517	0.755	0.756

Table 5: Values of the negative log-likelihood (the "loss") and accuracy averaged over the training samples or testing samples from data on forest cover type, with $\gamma = 10^5$ being the maximal possible value of the random variable distributed uniformly over $[-\gamma, \gamma]$ added to shares of the data; the identity link does not solve a classification problem, so has no natural notion of accuracy for this binary classification

generalization from the training set to the testing set. Figures 10 and 11 and Table 5 include in addition the results of using the identity for the link (needless to say, the identity link results in real numbers that need not be limited to the values 0 and 1 corresponding to classes 0 and 1 from the binarized data set — the identity link does not produce a classification, although classification is the natural task for this data). The identity link runs successfully, though of course the resulting negative log-likelihoods ("loss") have no natural interpretation.

Figure 12 displays the results of training multinomial logistic regression for all 7 classes, together with applying the resulting trained model to the testing set, as a function of the maximum γ of the random variable distributed uniformly over $[-\gamma, \gamma]$; the accuracy is excellent from $\gamma = 10^3$ to $\gamma = 10^6$. The generalization from the training set to the testing set is perfect. At $\gamma = 10^5$, the training loss is 0.705, the testing loss is 0.711, the training accuracy is 0.710, and the testing accuracy is 0.710; training in public produces the same results to within ±0.001.

5.2.3 Deaths from horsekicks

We use Poisson regression to analyze the classical data from Ladislaus Bortkiewicz tabulating the numbers of deaths from horsekicks in 14 corps of the Prussian army for each of the 20 years from 1875 to 1894, available (with extensive discussion) in the work of [19]. This data is a canonical example of counts which follow the Poisson distribution. We consider four separate Poisson regressions, for the following sets of covariates: (0) no covariates, (1) one-hot encodings of the corps, (2) second-order polynomials of the years, and (3) concatenating the one-hot encodings of the corps and the second-order polynomials of the years. The one-hot encoding of a corps is a vector with 14 entries, one of which is 1 and 13 of which are 0; the position of the entry that is 1 corresponds to the associated corps. The second-order polynomials of the years arise from using as covariate vector a vector with 3 entries, the entries being the constant 1, the number of years beyond 1875, and the square of the number of years beyond 1875; Poisson regression considers linear combinations of these entries, thus forming quadratic functions of the years.

The log-likelihoods of the fully trained models are the same to three-digit precision when comparing training in public to training in private; with 14 samples per minibatch (where 14 is the number of corps, so seemed like a natural choice), convergence of the log-likelihoods required 50,000 iterations (which is 2,500 epochs) with the second-order polynomials of the years in the covariates but only 10,000 iterations (500 epochs) without, all at a learning rate 2×10^{-2}. The negative log-likelihoods of the trained models for the four sets of covariates converge to (0) 1.124, (1) 1.077,

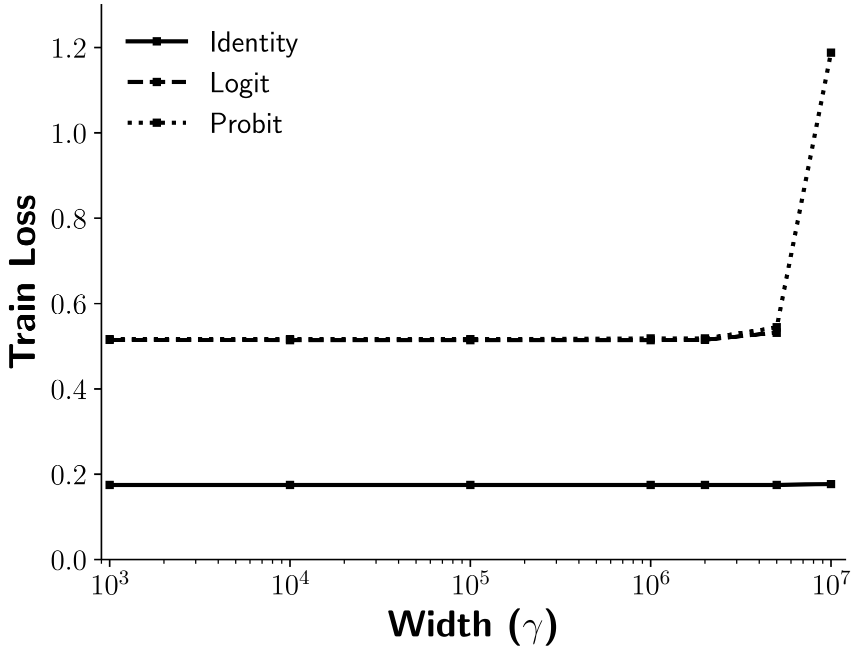

Figure 10: Value of the negative log-likelihood (the "loss") averaged over the training data on forest cover type, after convergence of the training iterations, as a function of the maximum value γ of the random variable distributed uniformly on $[-\gamma, \gamma]$ added to the shares of the data

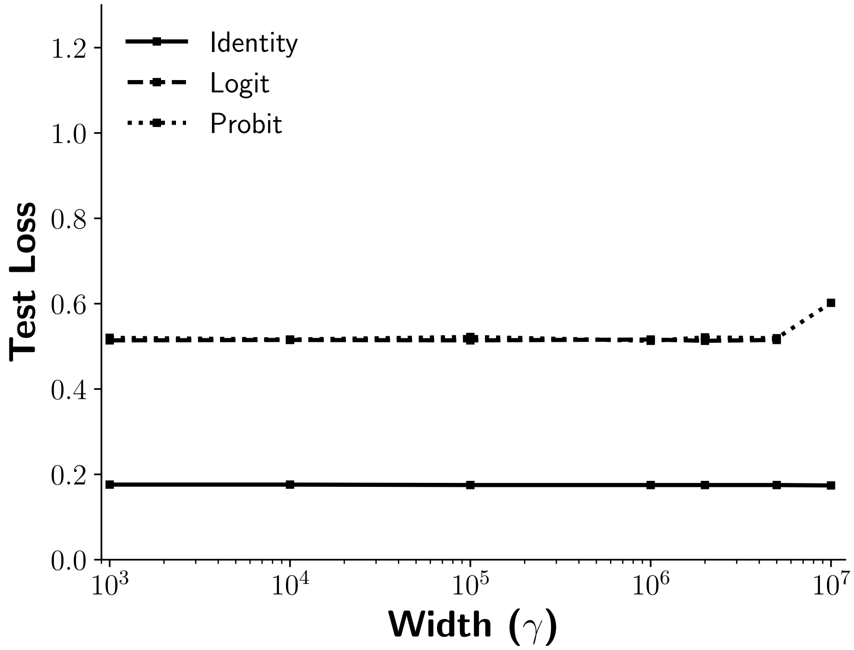

Figure 11: Value of the negative log-likelihood (the "loss") averaged over the testing data on forest cover type, as a function of the maximum value γ of the random variable distributed uniformly on $[-\gamma, \gamma]$ added to the shares of the data

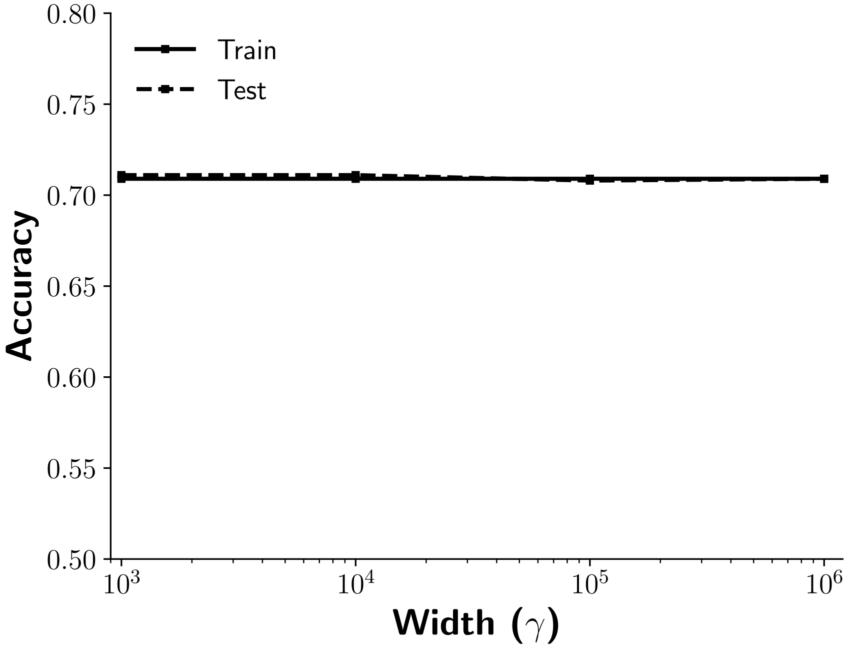

Figure 12: Accuracy of multinomial logistic regression averaged over the training or testing data on forest cover type, as a function of the maximum value γ of the random variable distributed uniformly on $[-\gamma, \gamma]$ added to the shares of the data

(2) 1.107, and (3) 1.061, averaged over all samples (there are 280 samples in total — 14 corps for each of 20 years). Twice the negative log-likelihood is sometimes called the "deviance," which is generally defined only up to an additive constant. The decrease in deviance when moving from covariates (0) to (1) is 26.3, the decrease from (0) to (2) is 9.5, and from (0) to (3) is 35.3; these deviances refer to the totals over all 280 samples, so are 280 times the average over the samples. The degrees of freedom corresponding to each of these changes is less than the decrease in deviance, indicating some minor heterogeneity in the data. (The baseline deviance generally has no statistical interpretation in terms of a universal distribution such as χ^2; changes in deviance when changing covariates are what matter.)

With 14 samples per minibatch, each iteration of stochastic gradient descent takes about 1.7×10^{-2} seconds when training in private for any of the four sets of covariates; the smallest set (0) takes about 0.5×10^{-2} seconds less per iteration than the largest set (3). When training in public, each iteration takes about 1.7×10^{-4} seconds for any of the four sets of covariates. Training in private thus takes about 100 times longer than training in public on standard machines in a cluster for software development at Facebook.

6 Discussion and conclusion

The scheme proposed in the present paper has several strengths. One is the simple relation between the machine precision and the best settings for parameters; for instance, our theoretical analysis points to $\gamma = 10^5$ for IEEE standard double-precision arithmetic, and indeed $\gamma = 10^5$ works well in all our numerical experiments. Naturally, this requires that the numbers being encrypted be appropriately normalized (so that β need not be too large), as usually achieved in modern machine learning via various normalizations and normalization layers of neural networks. For shallow networks (such as the generalized linear models considered above), such normalization is standard as preprocessing of the data. In principle, the present paper discusses the numerical tools required for other normalizations that are common in machine learning (such tools include raising to the power $-1/2$, as in the reciprocal of a Euclidean norm); however, handling deep networks automatically is likely to require further work beyond the scope of the present paper.

Another strength is that those who provide the sensitive data being processed need only provide their shares once and for all to the multiple parties, prior to the execution of any secure multiparty computations. Although the multiple parties must conspire together via communication among themselves in the all-reduce operations of the Beaver multiplication and squaring in Tables 1 and 2, there is no need for the data providers to participate in any of the processing (aside from providing the shares of data in the first place). Indeed, multiple data providers never need to communicate with each other, but only with the multiple parties that will process the data — and only once, before any computation commences. Please note that this strength is common to most schemes proposed for secure multiparty computations and is in no way unique to ours.

Undoubtedly the greatest strength is that the proposal of the present paper requires nothing but IEEE standard double-precision floating-point arithmetic, while still guaranteeing perfect privacy at a precision of about 10^{-5}, with all information leakage rigorously controlled courtesy of full mathematical proofs. We suspect that the traditional discrete secure multiparty computations might necessarily leak about as much information via side channels such as measurements of timing or usage of computational resources, when approximating floating-point calculations with discretizations. In any case, our scheme's requiring nothing more than the standard floating-point arithmetic greatly facilitates implementation, and conveniently allows for optimized performance by leveraging existing hardware and software. That said, many of the same benefits are available on secure hardware supporting the Intel SGX of [1], AMD SEV-SNP of [14], IBM Secure Execution of [17], and other such frameworks, at least on special secure microprocessors. We look forward to investigating combinations of our secure multiparty computations with such secure hardware, hoping to gain the advantages of both hardware- and software-based security.

A Proof of Theorem 1

A simple, standard computation yields that the differential entropy, in bits, of a random variable Y distributed uniformly over $[-\gamma, \gamma]$ is

$$h(Y) = \log_2(2\gamma). \tag{34}$$

Combining (1), (34), and the upper-bound on $h(X + Y)$ from the following lemma yields (2), completing the proof of Theorem 1.

Lemma 6. *Suppose that X and Y are independent scalar random variables and β and γ are positive real numbers such that $|X| \le \beta < \gamma$ and Y is distributed uniformly over $[-\gamma, \gamma]$. Then,*

$$h(X + Y) \le \frac{\beta}{\gamma} + \log_2(2\gamma), \tag{35}$$

where h denotes the differential entropy measured in bits (not nats), with equality attained in (35) when X is β times a Rademacher variate, that is, when $X = \beta$ with probability $1/2$ and $X = -\beta$ with probability $1/2$.

Proof. Denoting the cumulative distribution function of X by F, the probability density function of $X+Y$ is

$$g(y) = \frac{1}{2\gamma}\int_{-\gamma}^{\gamma} dF(y-x) = \frac{1}{2\gamma}\int_{y-\gamma}^{y+\gamma} dF(x) = \frac{F(y+\gamma)-F(y-\gamma)}{2\gamma}. \tag{36}$$

Combining $|X| \le \beta$ and the definition of a cumulative distribution function yields that $F(x) = 0$ for $x < -\beta$ and $F(x) = 1$ for $x > \beta$, so (36) becomes (as diagrammed in Figure 13)

$$g(y) = \begin{cases} 0, & y < -\gamma-\beta \\ \frac{F(y+\gamma)}{2\gamma}, & -\gamma-\beta < y < -\gamma+\beta \\ \frac{1}{2\gamma}, & -\gamma+\beta < y < \gamma-\beta \\ \frac{1-F(y-\gamma)}{2\gamma}, & \gamma-\beta < y < \gamma+\beta \\ 0, & y > \gamma+\beta \end{cases} \tag{37}$$

and the entropy in bits of $X+Y$ is

$$\begin{aligned} h(X+Y) &= -\int_{-\infty}^{\infty} g(y)\,\log_2(g(y))\,dy \\ &= -\frac{1}{2\gamma}\int_{\gamma-\beta}^{\gamma+\beta}\left(F(\gamma-y)\,\log_2\left(\frac{F(\gamma-y)}{2\gamma}\right) + (1-F(y-\gamma))\,\log_2\left(\frac{1-F(y-\gamma)}{2\gamma}\right)\right)dy \\ &\qquad -\frac{1}{2\gamma}\int_{-\gamma+\beta}^{\gamma-\beta}\log_2\left(\frac{1}{2\gamma}\right)dy \\ &= -\frac{1}{2\gamma}\int_{-\beta}^{\beta}\Big(F(-y)\,\log_2(F(-y)) + (1-F(y))\,\log_2(1-F(y))\Big)\,dy + \log_2(2\gamma) \\ &= -\frac{1}{2\gamma}\int_{-\beta}^{\beta}\Big(F(y)\,\log_2(F(y)) + (1-F(y))\,\log_2(1-F(y))\Big)\,dy + \log_2(2\gamma). \end{aligned} \tag{38}$$

The function $p\log_2(p)+(1-p)\log_2(1-p)$ is minimal for $0 < p < 1$ at $p = 1/2$, so the integral in the right-hand side of (38) is minimal when $F(y) = 1/2$ for $|y| < \beta$, in which case (38) becomes (35) with equality attained when X is β times a Rademacher variate. □

B Proof of Theorem 2

For any scalar random variables X, Y, and Z, the definition of mutual information states

$$I(X;\,X+Y,\,X+Z) = h(X+Y,\,X+Z) - h(X+Y,\,X+Z \mid X), \tag{39}$$

$$I(X;\,X+Y) = h(X+Y) - h(X+Y \mid X), \tag{40}$$

and

$$I(X;\,X+Z) = h(X+Z) - h(X+Z \mid X), \tag{41}$$

where h denotes entropy. Furthermore, the subadditivity of entropy for any arbitrary random variables yields

$$h(X+Y,\,X+Z) \le h(X+Y) + h(X+Z). \tag{42}$$

Taking X, Y, and Z to be independent then yields

$$\begin{aligned} h(X+Y,\,X+Z \mid X) &= h(Y,\,Z \mid X) = h(Y,\,Z) = h(Y) + h(Z) \\ &= h(Y \mid X) + h(Z \mid X) = h(X+Y \mid X) + h(X+Z \mid X). \end{aligned} \tag{43}$$

Combining (39)–(43) yields (3), completing the proof of Theorem 2.

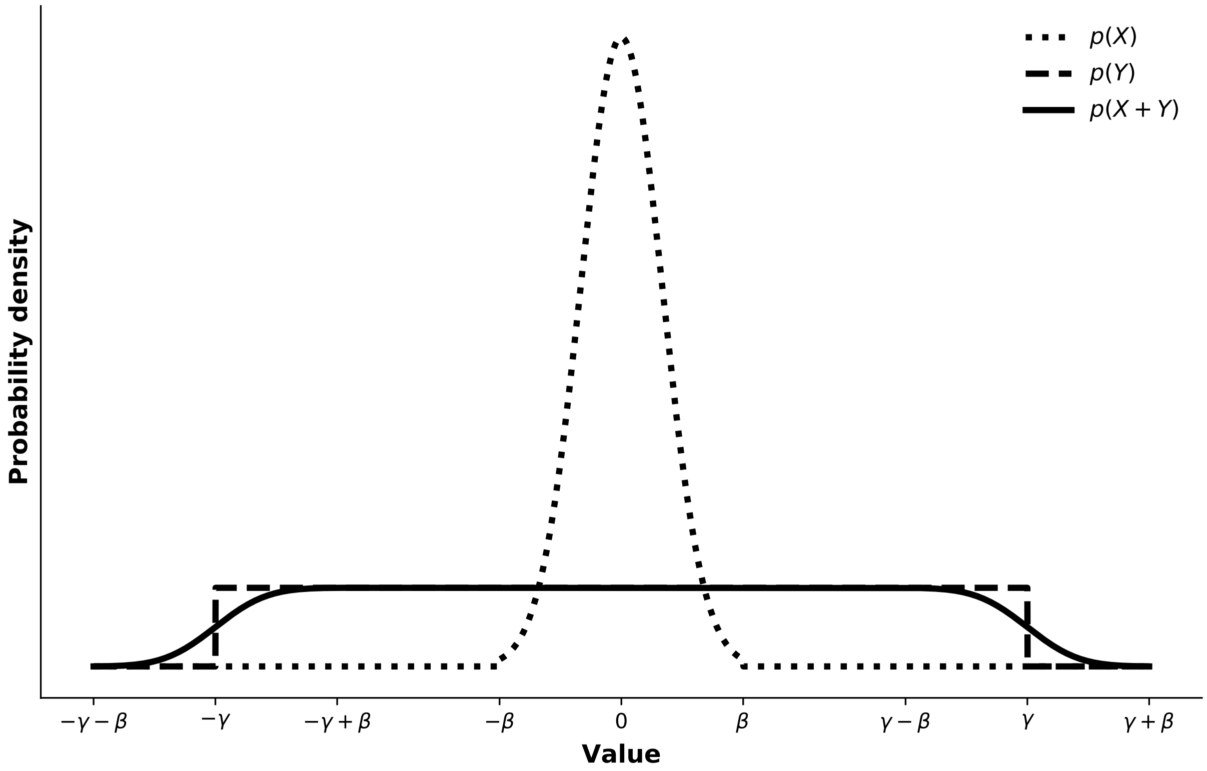

Figure 13: Sketch of a visualization for understanding the proof of Lemma 6, in the case where the probability distribution of X is given by an even unimodal probability density

C Proof of Theorem 3

We suppose that X, Y, P, Q, and T are independent scalar random variables and γ is a positive real number such that $|X| \le 1 < 3 < \gamma$, the random variable T is distributed uniformly over $[-\gamma^2, \gamma^2]$, and Y, P, and Q are distributed uniformly over $[-\gamma, \gamma]$. Then, since $(X - Y) - (P - Q)$ is just the difference between $X - Y$ and $P - Q$, providing no more and no less information than $X - Y$ and $P - Q$ on their own without $(X - Y) - (P - Q)$, and $(P^2 - T) + 2(P - Q)(X - P) + (X - P)^2/2$ is also merely a deterministic combination of the observations $P^2 - T$, $P - Q$, and $X - P$, providing no more and no less information than $P^2 - T$, $P - Q$, and $X - P$ on their own without $(P^2 - T) + 2(P - Q)(X - P) + (X - P)^2/2$, the mutual information satisfies

$$I\Big(X;\, X-Y,\, P-Q,\, P^2-T,\, (X-Y)-(P-Q),\, X-P,\, (P^2-T)+2(P-Q)(X-P)+(X-P)^2/2\Big) \\ = I(X;\, X-Y,\, P-Q,\, P^2-T,\, X-P). \quad (44)$$

Since $X - Q = (X - P) + (P - Q)$ and $P - Q = (X - Q) - (X - P)$ establish a bijection between the pair $X - P$ and $X - Q$ and the pair $X - P$ and $P - Q$, it follows that

$$I(X;\, X-Y,\, P-Q,\, P^2-T,\, X-P) = I(X;\, X-Y,\, X-Q,\, X-P,\, P^2-T). \quad (45)$$

Combining (3) and the fact that X, Y, Q, P, and T are independent yields

$$I(X;\, X-Y,\, X-Q,\, X-P,\, P^2-T) \le I(X;\, X-Y) + I(X;\, X-Q) + I(X;\, X-P,\, P^2-T). \quad (46)$$

Combining (44)–(46) and (47) from the following lemma, together with (2), yields (4), completing the proof of Theorem 3.

Lemma 7. *Suppose that X, P, and T are independent scalar random variables and γ is a positive real number such that $|X| \le 1 < 3 < \gamma$, the random variable P is distributed uniformly over $[-\gamma, \gamma]$, and T is distributed uniformly over $[-\gamma^2, \gamma^2]$. Then,*

$$I(X;\, X-P,\, P^2-T) \le I(X;\, X-P) + \frac{2}{\gamma} + \frac{1}{\gamma^2}, \quad (47)$$

where I denotes the mutual information measured in bits.

Proof. The proof begins with a string of identities, systematically simplifying (or re-expressing) their right-hand sides. Indeed, since $X^2 - 2XP + P^2$ is simply the square of $X - P$, it follows that

$$I(X;\, X-P,\, P^2-T) = I(X;\, X-P,\, X^2-2XP+P^2,\, P^2-T). \quad (48)$$

Since $X^2 - 2XP + T = (X^2 - 2XP + P^2) - (P^2 - T)$ and $P^2 - T = (X^2 - 2XP + P^2) - (X^2 - 2XP + T)$ establish a bijection between the pair $X^2 - 2XP + P^2$ and $P^2 - T$ and the pair $X^2 - 2XP + P^2$ and $X^2 - 2XP + T$, it follows that

$$I(X;\, X-P,\, X^2-2XP+P^2,\, P^2-T) = I(X;\, X-P,\, X^2-2XP+P^2,\, X^2-2XP+T). \quad (49)$$

Since $X^2 - 2XP + P^2$ is simply the square of $X - P$, it follows that

$$I(X;\, X-P,\, X^2-2XP+P^2,\, X^2-2XP+T) = I(X;\, X-P,\, X^2-2XP+T). \quad (50)$$

The chain rule for mutual information yields

$$I(X;\, X-P,\, X^2-2XP+T) = I(X;\, X-P) + I(X;\, X^2-2XP+T \mid X-P). \quad (51)$$

The definition of mutual information states

$$\begin{aligned}&I(X;\, X^2 - 2XP + T \mid X - P)\\ &\qquad = h(X^2 - 2XP + T \mid X - P) - h(X^2 - 2XP + T \mid X - P,\, X), \quad (52)\end{aligned}$$

where h denotes the differential entropy measured in bits. Since T is independent of X and P, the last term in the right-hand side of (52) is

$$h(X^2 - 2XP + T \mid X - P,\, X) = h(X^2 - 2XP + T \mid P, X) = h(T). \quad (53)$$

The fact that T is distributed uniformly over $[-\gamma^2, \gamma^2]$ yields via a simple, straightforward calculation that

$$h(T) = \log_2(2\gamma^2). \quad (54)$$

We now upper-bound the first term in the right-hand side of (52), by defining

$$S = X^2 - 2XP \quad (55)$$

and noticing

$$|S| \leq |X|^2 + 2|X||P| \leq 1 + 2\gamma. \quad (56)$$

That conditioning never increases entropy yields that the first term in the right-hand side of (52) satisfies

$$h(S + T \mid X - P) \leq h(S + T). \quad (57)$$

Combining (35) and the fact that T is distributed uniformly over $[-\gamma^2, \gamma^2]$ yields that, maximizing over all random variables S such that $|S| \leq \beta = 1 + 2\gamma$ (even dropping the constraint that $S = X^2 - 2XP$ in the maximization),

$$h(S + T) \leq \frac{1 + 2\gamma}{\gamma^2} + \log_2(2\gamma^2) = \frac{2}{\gamma} + \frac{1}{\gamma^2} + \log_2(2\gamma^2). \quad (58)$$

Combining (52)–(58) yields that, maximizing over any random variable X such that $|X| \leq 1$,

$$I(X;\, X^2 - 2XP + T \mid X - P) \leq \frac{2}{\gamma} + \frac{1}{\gamma^2}. \quad (59)$$

Combining (48)–(51) and (59) yields (47). □

D Chebyshev series for odd functions

In this appendix, we review the approximation of odd functions via Chebyshev series, as summarized for general (not necessarily odd) functions in Sections 4–6 of [8].

Given a real-valued differentiable function f on $[-z, z]$ that is odd, that is,

$$f(-x) = -f(x) \quad (60)$$

for any real number x such that $-z \leq x \leq z$, we can approximate f via its Chebyshev series, as follows. First, we select a sufficiently large positive integer n and compute

$$c_j = \frac{2}{n} \sum_{k=1}^{n} \cos\left(\frac{j(2k-1)\pi}{4n}\right)\, f\left(z \cos\left(\frac{(2k-1)\pi}{4n}\right)\right) \quad (61)$$

for $j = 1, 3, \ldots, 2n-1$; the approximation will converge as n increases. Having calculated c_1, c_3, $\ldots$, c_{2n-1} from (61), we can compute a good approximation to f evaluated at any real number y such that $-z \le y \le z$:

$$f(y) \approx \sum_{j=1}^{n} c_{2j-1}\, T_{2j-1}(y/z), \tag{62}$$

where $T_j(x)$ denotes the Chebyshev polynomial of degree j evaluated at $x = y/z$.

To calculate the right-hand side of (62) efficiently using only additions and multiplications involving the input

$$x = y/z, \tag{63}$$

we evaluate the sums

$$s_{2k-1} = \sum_{j=1}^{k} c_{2j-1}\, T_{2j-1}(x) \tag{64}$$

for $k = 1, 2, \ldots, n$, via the following recurrence:

$$t_{2k+1} = (4x^2 - 2)t_{2k-1} - t_{2k-3} \tag{65}$$

and

$$s_{2k+1} = s_{2k-1} + c_{2k+1}t_{2k+1}, \tag{66}$$

started with

$$t_1 = x, \tag{67}$$

$$t_3 = (4x^2 - 3)x, \tag{68}$$

$$s_1 = c_1 t_1, \tag{69}$$

and

$$s_3 = s_1 + c_3 t_3. \tag{70}$$

The final sum s_{2n-1} is equal to the right-hand side of (62), due to (63) and (64).

E Review of stochastic gradient descent with minibatches

As discussed in any standard reference on modern machine learning, such as that of [4], minibatched stochastic gradient descent (SGD) calculates a vector w of parameters that minimizes the expected value $\mathbf{E}(\ell(X;w))$, where ℓ is a function of both w and a random vector X. The expected value is known as the "risk" and ℓ is known as the "loss." SGD minimizes the expected loss without having direct access to the probability distribution of X, instead relying solely on samples of X (usually drawn at random from a so-called "training set"). Minibatched SGD generates a sequence of approximations via iterations,

$$w^{(k+1)} = w^{(k)} - \frac{\eta}{m} \sum_{j=1}^{m} \frac{\partial}{\partial w} \ell(x^{(j,k)}; w) \bigg|_{w=w^{(k)}}, \tag{71}$$

where η is a positive real number known as the "learning rate," $\partial/\partial w$ denotes the gradient with respect to w (which is ℓ's second argument), m is the number of samples in a so-called "minibatch," and $x^{(1,k)}$, $x^{(2,k)}$, $\ldots$, $x^{(m,k)}$ denote samples from X. The iterations fail to minimize the expected loss when η is constant rather than decaying to 0 as the iterations proceed, but fixing η at a sensibly small value is a common practice in machine learning (and still ensures convergence to the minimum of the empirical risk under suitable conditions on ℓ, that is, to the minimum of the average of the loss, averaged over all samples in a fixed, finite training set).

Minimizing the regularized objective function $\mathbf{E}(\ell(X; w)) + \rho\|w\|_2^2/2$, where ρ is a nonnegative real number and $\|w\|_2$ denotes the Euclidean norm of w, is a common way of ensuring that w not become too large. Adding such regularization to SGD is also known as "weight decay," and the iterations in (71) become

$$w^{(k+1)} = w^{(k)} - \frac{\eta}{m} \sum_{j=1}^{m} \frac{\partial}{\partial w} \ell(x^{(j,k)}; w) \,\bigg|_{w=w^{(k)}} - \; \eta\rho w^{(k)}; \tag{72}$$

we used some weight decay for the multinomial logistic regression of the measured data in Subsection 5.2 (but used no weight decay for any other results reported above, nor did we use any weight decay for iterative updates to the so-called bias offsets in c from the following appendix).

F Review of generalized linear models

As discussed in any standard reference on generalized linear models, such as that of [18], a generalized linear model regresses a random vector Y of so-called targets against a matrix X of so-called covariates via the model

$$g(\mathbf{E}(Y|X)) = Xw + c, \tag{73}$$

where g is known as the "link function," $\mathbf{E}(Y|X)$ is the conditional expectation of the vector Y of targets given X (the matrix of covariates), w is a vector of so-called "weights" (or "parameters"), and c is a vector whose entries are independent of the values of X, known as "biases." Table 6 lists several special cases of generalized linear models. Given independent samples of pairs $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, ..., $(x^{(n)}, y^{(n)})$, the standard method of fitting the vectors w of weights and c of biases is to minimize the negative of the natural logarithm of the likelihood, that is, to minimize the empirical risk $-\frac{1}{n}\sum_{k=1}^{n} \ln(p(y^{(k)}|x^{(k)}; w, c))$, where $p(y^{(k)}|x^{(k)}; w, c)$ denotes the probability (at the parameter values w and c) of observing $y^{(k)}$ given $x^{(k)}$. Minimizing $-\mathbf{E}(\ln(p(Y|X; w, c)))$ via the minibatched stochastic gradient descent of the previous appendix is another (nearly equivalent) approach.

The probability distribution of Y given X for all of the generalized linear models considered in Table 6, except for probit regression, is a so-called "exponential family" of the form

$$p(y|x; w, c) = f(y) \exp(y^\top xw + y^\top c - \psi(xw + c)), \tag{74}$$

where f is a nonnegative-valued function, and ψ is known as the "log partition function": $\psi(\theta) = \|\theta\|_2^2/2$ for the normal distribution, $\psi(\theta) = \ln(1+\exp(\theta))$ for the Bernoulli distribution, and $\psi(\theta) = \exp(\theta)$ for the Poisson distribution. For all these cases, substituting (74) and taking the gradient with respect to θ of both sides of $\int p(y|x; w, c)\, dy = 1$ yields after a straightforward calculation that $\partial\psi/\partial\theta = \mathbf{E}(Y|x)$, where $\theta = xw + c$; combined with (73) this yields that $g(\partial\psi/\partial\theta) = \theta$, so the link g is the inverse of $\partial\psi/\partial\theta$.

Combining (74) and the chain rule yields that the gradient with respect to w of $\ln(p(y|x; w, c))$ is

$$\frac{\partial}{\partial w} \ln(p(y|x; w, c)) = x^\top \left(y - \frac{\partial\psi}{\partial\theta} \,\bigg|_{\theta=xw+c} \right) \tag{75}$$

and the gradient with respect to c of $\ln(p(y|x; w, c))$ is

$$\frac{\partial}{\partial c} \ln(p(y|x; w, c)) = y - \frac{\partial\psi}{\partial\theta} \,\bigg|_{\theta=xw+c}. \tag{76}$$

name	distribution	name of link	link $g(\mu)$	range of mean μ
linear least squares	$N(\mu, I)$	identity	μ	all real vectors
logistic regression	Bernoulli	logit	$\ln(\mu/(1-\mu))$	unit interval $[0, 1]$
probit regression	Bernoulli	probit	$\Phi^{-1}(\mu)$	unit interval $[0, 1]$
Poisson regression	Poisson	log	$\ln(\mu)$	nonnegative real numbers
multinomial logistic regression	multinomial	log performed entry-by-entry	ln of each entry of μ	prob. simplex $\sum_{j=1}^{k} \mu_j = 1$ and $\mu_1, \mu_2, \dots, \mu_k \ge 0$

Table 6: Special cases of generalized linear models, where $N(\mu, I)$ denotes the (possibly multivariate) normal distribution with mean μ and variance-covariance matrix being the identity matrix I, and Φ is the cumulative distribution function for the standard normal distribution (so Φ^{-1} is the corresponding quantile function, the inverse of Φ); "linear least squares" is also known as "ordinary least squares" or the "general" linear model — a special case of the "generalized" linear model

Thus, the stochastic gradient descent of the previous appendix requires nothing more than addition, matrix-vector multiplications, and evaluation of the inverse $(\partial\psi/\partial\theta)$ of the link g. Needless to say, the inverse of the identity function is the identity function, the inverse of ln is exp, and the inverse of the inverse of the cumulative distribution function Φ for the standard normal distribution is Φ. A simple calculation shows that the inverse of the logit function $g(\mu) = \ln(\mu/(1-\mu))$ is the standard logistic function $\partial\psi/\partial\theta = 1/(1 + \exp(-\theta))$ and that the softmax detailed in Subsection 4.3 above inverts ln applied entrywise to a probability vector (the softmax simply applies exp entrywise and then normalizes to form a proper probability distribution).

The probability distribution for probit regression is an exponential family, but not of the form in (74); we handle this special case as detailed in Sub-subsection 5.1.3.

Acknowledgements

We would like to thank Shubho Sengupta, Andrew Tulloch, and the anonymous reviewers.

References

[1] Anati, I., Gueron, S., Johnson, S. P. & Scarlata, V. R. (2013) Innovative technology for CPU based attestation and sealing. in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ed. by R. Lee, & W. Shi, vol. 13, pp. 1–7. Association for Computing Machinery.

[2] Beaver, D. (1991) Efficient multiparty protocols using circuit randomization. in *Advances in Cryptology — Proceedings of the 1991 Annual International Cryptology Conference*, ed. by J. Feigenbaum, vol. 576 of *Lecture Notes in Computer Science*, pp. 420–432. Springer-Verlag.

[3] Bogdanov, D., Laur, S. & Willemson, J. (2008) Sharemind: a framework for fast privacy-preserving computations. in *Computer Security — Proceedings of the 2008 European Symposium on Research in Computer Security*, ed. by S. Jajodia, & J. Lopez, vol. 5283 of *Lecture Notes in Computer Science*, pp. 192–206. Springer-Verlag.

[4] Bottou, L., Curtis, F. E. & Nocedal, J. (2018) Optimization methods for large-scale machine learning. *SIAM Rev.*, **60**(2), 223–311.

[5] Boura, C., Chillotti, I., Gama, N., Jetchev, D., Peceny, S. & Petric, A. (2018) High-precision privacy-preserving real-valued function evaluation. in *Financial Cryptography and Data Security: 22nd International Conference, Nieuwpoort, Curaçao*, ed. by S. Meiklejohn, & K. Sako, vol. 10957 of *Lecture Notes in Computer Science*, pp. 183–202. Springer.

[6] Chang, C.-C. & Lin, C.-J. (2011) LIBSVM: a library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, **2**(3), 1–27, Article no. 27.

[7] Cover, T. M. & Thomas, J. A. (2006) *Elements of Information Theory*. Wiley-Interscience, 2nd edn.

[8] Curry, B. (2006) Parameter redundancy in neural networks: an application of Chebyshev polynomials. *Comput. Management Sci.*, **4**(3), 227–242.

[9] Dua, D. & Graff, C. (2019) UCI Machine Learning Repository. UC-Irvine, School of Information and Computer Sciences, Available at `http://archive.ics.uci.edu/ml`.

[10] Gunning, D., Hannun, A., Ibrahim, M., Knott, B., van der Maaten, L., Reis, V., Sengupta, S., Venkataraman, S. & Zhou, X. (2019) CrypTen: a new research tool for secure machine learning with PyTorch. Blog post available at `https://ai.facebook.com/blog/crypten-a-new-research-tool-for-secure-machine-learning-with-pytorch`.

[11] Guo, C.-H. & Higham, N. J. (2006) A Schur-Newton method for the matrix pth root and its inverse. *SIAM J. Matrix Anal. Appl.*, **28**(3), 788–804.

[12] Higham, N. J. (2005) The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, **26**(4), 1179–1193.

[13] IEEE Computer Society Microprocessor Standards Committee (2019) IEEE Standard for Floating-Point Arithmetic. Discussion Paper 754-2019, IEEE.

[14] Kaplan, D. (2019) Upcoming x86 technologies for malicious hypervisor protection. Linux Security Summit. Slides and recording available at `https://sched.co/TynP`.

[15] Kenney, C. & Laub, A. J. (1991) Rational iterative methods for the matrix sign function. *SIAM J. Matrix Anal. Appl.*, **12**(2), 273–291.

[16] LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M. & Huang, F. J. (2006) A tutorial on energy-based learning. in *Predicting Structured Data*, ed. by G. Bakir, T. Hofmann, B. Schölkopf, A. J. Smola, B. Taskar, & S. V. N. Vishwanathan, Neural Information Processing, pp. 191–246. MIT Press.

[17] Mauri, R. (2020) IBM Z deepens data privacy capabilities with new air-cooled models and IBM Secure Execution for Linux. Blog post available at `https://ibm.com/blogs/systems/secure-z-linuxone`.

[18] McCullagh, P. & Nelder, J. A. (1989) *Generalized Linear Models*, vol. 37 of *CRC Monographs on Statistics and Applied Probability*. Chapman and Hall, 2nd edn.

[19] Stigler, S. M. (2019) Data have a limited shelf life. *Harvard Data Science Review*, **1**(2), 1–23, Available at `https://hdsr.mitpress.mit.edu/pub/iu26pfw1`.