Training Models with End-to-End Low Precision: The Cans, the Cannots, and a Little Bit of Deep Learning

Hantian Zhang[†] Jerry Li[*] Kaan Kara[†]
Dan Alistarh[†] Ji Liu[‡] Ce Zhang[†]
[†] ETH Zurich
zht@student.ethz.ch
{kaan.kara, dan.alistarh, ce.zhang}@inf.ethz.ch
[‡] University of Rochester
jliu@cs.rochester.edu
[*] Massachusetts Institute of Technology
jerryzli@mit.edu

Abstract

¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ master Recently there has been significant interest in training machine-learning models at low precision: by reducing precision, one can reduce computation and communication by one order of magnitude. We examine training at reduced precision, both from a theoretical and practical perspective, and ask: *is it possible to* train *models at end-to-end low precision with* provable *guarantees? Can this lead to consistent order-of-magnitude speedups?*

¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ master For linear models, the answer is yes. We develop a simple framework based on one simple but novel strategy called double sampling. Our framework is able to execute training at low precision with no bias, guaranteeing convergence, whereas naive quantization would introduce significant bias. We validate our framework across a range of applications, and show that it enables an FPGA prototype that is up to $6.5\times$ faster than an implementation using full 32-bit precision. ¡¡¡¡¡¡¡ HEAD

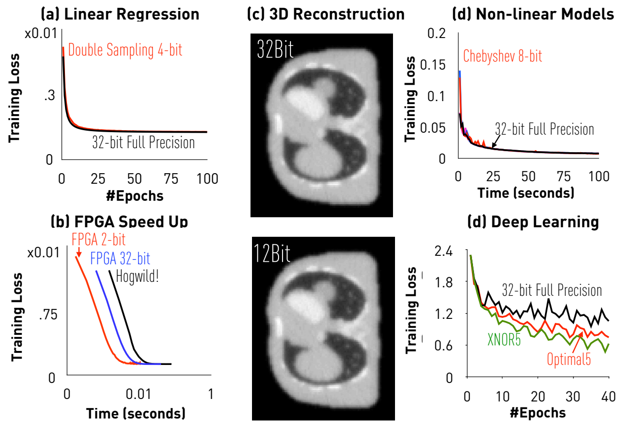
======= ¿¿¿¿¿¿¿ master We further develop a variance-optimal stochastic quantization strategy and show that it can make a significant difference in a variety of settings. When applied to linear models together with double sampling, we save up to another ¡¡¡¡¡¡¡ HEAD $1.7\times$ in data movement compared with uniform quantization. When training deep networks with quantized models, we achieve higher accuracy than the state-of-the-art XNOR-Net. ======= $1.7\times$ in data movement compared with the uniform quantization. When training deep networks with quantized models, we achieve higher accuracy than the state-of-the-art XNOR-Net. Finally, we extend our framework through approximation to non-linear models, such as SVM. We show that, although using low-precision data induces bias, we can appropriately bound and control the bias. We find in practice *8-bit* precision is often sufficient to converge to the correct solution. Interestingly, however, in practice we notice that our framework does not always outperform the naive rounding approach. We discuss this negative result in detail. ¿¿¿¿¿¿¿ master

=======
¿¿¿¿¿¿¿ master

1 Introduction

The computational cost and power consumption of today's machine learning systems are often driven by data movement, and by the precision of computation. In our experience, in applications such as tomographic reconstruction, anomaly detection in mobile sensor networks, and compressive sensing, the overhead of transmitting the data samples can be massive, and hence performance can hinge on reducing the precision of data representation and associated computation. A similar trend is observed in deep learning, where impressive progress has been reported with systems using end-to-end reduced-precision representations **????**. The empirical success of these works inspired this paper, in

Machine Learning Models	Data Movement Channels			Speed up because of our techniques
	Gradient	**Input Samples**	**Model**	
Linear Models	De Sa et la., Alistarh et al., ...	🟩 1. Double Sampling 2. Data-Optimal Encoding	🟩 Stochastic Rounding	🟩 Very Significant Speed up (Up to 10x)
Generalized Linear Models		🟥 1. Chebyshev Approximation 2. Refetching Heuristics for SVM		🟥 Minor/No Speed up
Deep Learning		Courbariaux et al., Rastegari et al., ...	🟩 Data-Optimal Encoding	🟩 Significant Speed up



¡¡¡¡¡¡¡ HEAD

Figure 1: Overview of theoretical results and highlights of empirical results. See Introduction for details.

which we try to provide a *theoretical* understanding of end-to-end low-precision training for machine learning models. In this context, the motivating question behind our work is: *When training general machine learning models, can we lower the precision of data representation, communication, and computation, while maintaining provable guarantees?*

In this paper, we develop a general framework to answer this question, and present both positive and negative results obtained in the context of this framework. Figure 1 encapsulates our results: (a) for linear models, we are able to lower the precision of both computation and communication, including input samples, gradients, and model, by up to 16 times, while still providing rigorous theoretical guarantees; (b) our FPGA implementation of this framework achieves up to $6.5\times$ speedup compared with a 32-bit FPGA implementation, or with a 10-core CPU running Hogwild!; (c) we are able to decrease data movement by $2.7\times$ for tomographic reconstruction, while obtaining a negligible quality decrease. Elements of our framework generalize to (d) model compression for training deep learning models. In the following, we describe our technical contributions in more detail.

¡¡¡¡¡¡¡ HEAD

1.1 Summary of Technical Contributions

=======

1.2 Summary of Technical Contributions

¿¿¿¿¿¿¿ master

We consider the following problem in training generalized linear models:

$$\min_{\mathbf{x}} : \quad \frac{1}{2K}\sum_{k=1}^{K} l(\mathbf{a}_k^\top \mathbf{x}, b_k)^2 + R(\mathbf{x}), \tag{1}$$

where $l(\cdot,\cdot)$ is a loss function and R is a regularization term that could be ℓ_1 norm, ℓ_2 norm, or even an indicator function representing the constraint. The gradient at the sample $(\mathbf{a}_k, b_k)$ is:

$$\mathbf{g}_k := \mathbf{a}_k \frac{\partial l(\mathbf{a}_k^\top \mathbf{x}, b_k)}{\partial \mathbf{a}_k^\top \mathbf{x}}.$$

We denote the problem dimension by n. We consider the properties of the algorithm when a lossy compression scheme is applied to the data (samples), gradient, and model, to reduce the communication cost of the algorithm—that is, we consider quantization functions Q_g, Q_m, and Q_s for gradient, model, and samples, respectively, in the gradient update:

$$\mathbf{x}_{t+1} \leftarrow \text{prox}_{\gamma R(\cdot)} \left(\mathbf{x}_t - \gamma Q_g(\mathbf{g}_k(Q_m(\mathbf{x}_t), Q_s(\mathbf{a}_t))) \right), \tag{2}$$

where the proximal operator is defined as

$$\text{prox}_{\gamma R(\cdot)}(\mathbf{y}) = \underset{\mathbf{x}}{\text{argmin}} \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2 + \gamma R(\mathbf{x}).$$

¡¡¡¡¡¡¡ HEAD

Our Results We summarize our results as follows.

Linear Models. When $l(\cdot, \cdot)$ is =======

Our Results We summarize our results as follows. The **(+)** sign denotes a “positive result,” where we achieve significant practical speedup; it is **(–)** otherwise.

(+) Linear Models. When $l(\cdot, \cdot)$ is ¿¿¿¿¿¿¿ master the least squares loss, we first notice that simply doing stochastic quantization of data samples (i.e., Q_s) introduces bias of the gradient estimator and therefore SGD would converge to a different solution. We propose a simple solution to this problem by introducing a *double sampling* strategy $\tilde{Q}_s$ that uses multiple samples to eliminate the correlation of samples introduced by the non-linearity of the gradient. We analyze the additional variance introduced by double sampling, and find that its impact is *negligible in terms of convergence time* as long as the number of bits used to store a quantized sample is at least $\Theta(\log n/\sigma)$, where σ^2 is the variance of the standard stochastic gradient. This implies that the 32-bit precision may be excessive for many practical scenarios.

¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ master We build on this result to obtain an *end-to-end quantization* strategy for linear models, which compresses all data movements. For certain settings of parameters, end-to-end quantization adds as little as a *constant factor* to the variance of the entire process.

¡¡¡¡¡¡¡ HEAD

Optimal Quantization and Extension to Deep Learning. =======

(+) Optimal Quantization and Extension to Deep Learning. ¿¿¿¿¿¿¿ master We then focus on reducing the variance of stochastic quantization. We notice that different methods for setting the quantization points have different variances—the standard uniformly-distributed quantization strategy is far from optimal in many settings. We formulate this as an independent optimization problem, and solve it optimally with an efficient dynamic programming algorithm that only needs to scan the data in a single pass. When applied to linear models, this optimal strategy can save up to $1.6\times$ communication compared with the uniform strategy.

¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ master We perform an analysis of the optimal quantizations for various settings, and observe that the uniform quantization approach popularly used by state-of-the-art end-to-end low-precision deep learning training systems when more than 1 bit is used is suboptimal. We apply optimal quantization to models and show that, with one standard neural network, we outperform the uniform quantization used by XNOR-Net and a range of other recent approaches. This is related, but different, to recent work on model compression for inference **?**. To the best of our knowledge, this is the first time such optimal quantization strategies have been applied to training.

¡¡¡¡¡¡¡ HEAD

	Regression		
Dataset	Training Set	Testing Set	# Features
Synthetic 10	10,000	10,000	10
Synthetic 100	10,000	10,000	100
Synthetic 1000	10,000	10,000	1,000
YearPrediction	463,715	51,630	90
cadata	10,000	10,640	8
cpusmall	6,000	2,192	12
	Classification		
Dataset	Training Set	Testing Set	# Features
cod-rna	59,535	271,617	8
gisette	6,000	1,000	5,000
	Deep Learning		
Dataset	Training Set	Testing Set	# Features
CIFAR-10	50,000	10,000	$32 \times 32 \times 3$
	Tomographic Reconstruction		
Dataset	# Projections	Volumn Size	Proj. Size
	128	128^3	128^3

¡¡¡¡¡¡¡ HEAD

Table 1: Dataset statistics

8 Experiments

We provide empirical validation of our framework.
¿¿¿¿¿¿¿ master

Experimental Setup Table 6 shows the datasets we use. Unless otherwise noted, we always use diminishing step-sizes α/k, where k is the current number of epoch. We tune α for the full precision implementation, and use the same initial step size for our low-precision implementation. (Theory and experiments imply that the low-precision implementation often favors smaller step size. Thus we do not tune step sizes for the low-precision implementation, as this can only improve the accuracy of our approach.)
¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ master

Summary of Experiments Due to space limitations, we only report on **Synthetic 100** for regression, and on **gisette** for classification. The additional material contains (1) several other datasets, and discusses (2) different factors such as impact of the number of features, (3) FPGA implementation and design decisions, and (4) refetching heuristics.

8.1 Linear Models

=======

8.2 Linear Models

¿¿¿¿¿¿¿ master
For linear models, we validate that (1) with double sampling, SGD with low precision converges—in comparable empirical convergence rates—to the same solution as SGD with full precision; and (2) implemented on FPGA, our low-precision prototype achieves significant speedup because of the decrease in bandwidth consumption.
¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ master

Convergence Figure 7 illustrates the result of training linear models: (a) linear regression and (b) least squares SVMs, with end-to-end low-precision and full precision. For low precision, we pick the smallest number of bits that results in a smooth convergence curve. We compare the final training loss in both settings and the convergence rate.

¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ master We see that, for both linear regression and least squares SVM, using 5- or 6-bit is always enough to converge to the same solution with comparable convergence rate. This validates our prediction that double-sampling provides an unbiased estimator of the gradient. Considering the size of input samples that we need to read, we could potentially save 6–8× memory bandwidth compared to using 32-bit.

¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ master

Speedup We implemented our low-precision framework on a state-of-the-art FPGA platform. We leave the detailed implementation to the full version. This implementation assumes the input data is already quantized and stored in memory (data can be quantized during the first epoch).

¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ master Figure 9 illustrates the result of (1) our FPGA implementation with quantized data, (2) FPGA implementation with 32-bit data, and (3) Hogwild! running with 10 CPU cores. Observe that all approaches converge to the same solution. FPGA with quantized data converges 6-7× faster than FPGA with full precision or Hogwild!. The FPGA implementation with full precision is memory-bandwidth bound, and by using our framework on quantized data, we save up to 8× memory-bandwidth, which explains the speedup.

8.3 Data-Optimal Quantization Strategy

=======

8.4 Data-Optimal Quantization Strategy

¿¿¿¿¿¿¿ master

We validate that, with our data-optimal quantization strategy, we can significantly decrease the number of bits that double-sampling requires to achieve the same convergence. Figure 10(a) illustrates the result of using 3-bit and 5-bit for uniform quantization and optimal quantization on the **YearPrediction** dataset. We see that, while uniform quantization needs 5-bit to converge smoothly, optimal quantization only needs 3-bit. We save almost 1.7× number of bits by just allocating quantization points carefully.

¡¡¡¡¡¡¡ HEAD

Comparision with uniform quantization We validate that, with our data-optimal quantization strategy, we can significantly increase the convergence speed.

Figure 15 illustrates the result of training linear regression models: with uniform quantization points and optimal quantization points. Here, notice that we only quantize data, but not gradient or model. We see that, if we use same number of bits, optimal quantization always converges faster than uniform quantization and the loss curve is more stable, because the variance induced by quantization is smaller. As a result, with our data-optimal quantization strategy We can either (1) get higher convergence speed with the same number of bits; or (2) use less bits while getting the same convergence speed.

We also see from Figure 15 (a) to (c) that if the dataset has more features, usually we need more bits for quantization, because the variance induced by quantization is higher when the dimensionality is higher.

8.5 Extensions to Deep Learning

=======

8.6 Extensions to Deep Learning

¿¿¿¿¿¿¿ master

We validate that our data-optimal quantization strategy can be used in training deep neural networks. We take Caffe's CIFAR-10 tutorial **?** and compare three different quantization strategies: (1) Full Precision, (2) XNOR5, a XNOR-Net implementation that, following the multi-bits strategy in the original paper, quantizes data into five uniform levels, and (3) Optimal5, our quantization strategy with five optimal quantization levels. As shown in Figure 10(b), Optimal5 converges to a significantly lower training loss compared with XNOR5. Also, Optimal5 achieves >5 points higher testing accuracy over XNOR5. This illustrates the improvement obtainable by training a neural network with a carefully chosen quantization strategy.

¡¡¡¡¡¡¡ HEAD

9 Related Work

=======

9.1 Non-Linear Models

We validate that (1) our Chebyshev approximation approach is able to converge to almost the same solution with 8-bit precision for both SVM and logistic regression; and (2) we are nevertheless able to construct a straw man with 8-bit deterministic rounding or naive stochastic rounding to achieve the same quality and convergence rate.

Chebyshev Approximations Figure 12 illustrates the result of training SVM and logistic regression with Chebyshev approximation. Here, we use Chebyshev polynomials up to degree 15 (which requires 16 samples that can be encoded with 4 extra bits). For each sample, the precision is 4-bit, and therefore, in total we use 8-bit for each single number in input samples. We see that, with our quantization framework, SGD converges to similar training loss with a comparable empirical convergence rate for both SVM and logistic regression. We also experience no loss in test accuracy.

Negative Results We are able to construct the following, much simpler strategy that also uses 8-bit to achieve the same quality and convergence rate as our Chebyshev. In practice, as both strategies incur bias on the result, we do *not* see strong reasons to use our Chebyshev approximation, thus we view this as a negative result. As shown in Figure 12, if we simply round the input samples to the nearest 8-bit fix point representation (or do rounding stochastically), we achieve the same, and sometimes better, convergence than our Chebyshev approximation.

10 Extended Experiments

This section provides more experiment results. All experiments settings are the same with the previous section.

10.1 Linear Models

For linear models, we validate that with double sampling, SGD with low precision converges—in comparable empirical convergence rates—to the same solution as SGD with full precision and we want to understand how many bits do we need to achieve it empirically and how it is related to the size of dataset.

Figure 13 and 14 illustrates the result of training linear models: linear regression and least squares SVMs, respectively, with end-to-end low-precision and full precision. For low precision, we pick the smallest number of bits that results in a smooth convergence curve. We compare the final training loss in both settings and the convergence rate.

We see that, for both linear regression and least squares SVM, on all our datasets, using 5- or 6-bit is always enough to converge to the same solution with comparable convergence rate. This validates our prediction that double-sampling provides an unbiased estimator of the gradient. Considering the size of input samples that we need to read, we could potentially save 6–8$\times$ memory bandwidth compared to using 32-bit.

We also see from Figure 13 (a) to (c) that if the dataset has more features, usually we need more bits for quantization, because the variance induced by quantization is higher when the dimensionality is higher.

10.2 Data-Optimal Quantization Strategy for linear models

We validate that, with our data-optimal quantization strategy, we can significantly increase the convergence speed.

Figure 15 illustrates the result of training linear regression models: with uniform quantization points and optimal quantization points. Here, notice that we only quantize data, but not gradient or model. We see that, if we use same number of bits, optimal quantization always converges faster than uniform quantization and the loss curve is more stable, because the variance induced by quantization is smaller. As a result, with our data-optimal quantization strategy We can either (1) get higher convergence speed with the same number of bits; or (2) use less bits while getting the same convergence speed.

10.3 Non-Linear Models

Figure 16 illustrates the result of training SVM with refetching heuristic. We see that, with our refetching heuristic, SGD converges to similar training loss with a comparable empirical convergence rate for SVM. If we increase the number of bits we use, we need to refetch less data and if we use 8-bit quantization, we only need to fetch about 6% of data. We also experience no loss in test accuracy.

11 Related Work

¿¿¿¿¿¿¿ master

There has been significant work on "low-precision SGD" **??**. These results provide theoretical guarantees only for quantized gradients. The model and input samples, on the other hand, are much more difficult to analyze because of the non-linearity. We focus on *end-to-end* quantization, for all components.

¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ master

Low-Precision Deep Learning. Low-precision training of deep neural networks has been studied intensively and many heuristics work well for a subset of networks. OneBit SGD **?** provides a gradient compression heuristic developed in the context of deep neural networks for speech recognition. There are successful applications of end-to-end quantization to training neural networks that result in little to no quality loss **??????**. They quantize weights, activations, and gradients to low precision (e.g., 1-bit) and revise the back-propagation algorithm to be aware of the quantization function. The empirical success of these works inspired this paper, in which we try to provide a *theoretical* understanding of end-to-end low-precision training for machine learning models. Another line of research concerns inference and model compression of a pre-trained model **???????**. In this paper, we focus on training and leave the study of inference for future work.

¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ master

Low-Precision Linear Models. Quantization is a fundamental topic studied by the DSP community, and there has been research on linear regression models in the presence of quantization error or other types of noise. For example, **?** studied compressive sensing with binary quantized measurement, and a two-stage algorithm was proposed to recover the sparse high-precision solution up to a scale factor. Also, the classic errors-in-variable model **?** could also be relevant if quantization is treated as a source of "error." In this paper, we scope ourselves to the context of stochastic gradient descent, and our insights go beyond simple linear models. For SVM the straw man approach can also be seen as a very simple case of kernel approximation **?**.

¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ master

Other Related Work. Precision of data representation is a key design decision for configurable hardwares such as FPGA. There have been attempts to lower the precision when training on such hardware **?**. These results are mostly empirical; we aim at providing a theoretical understanding, which enables new algorithms.

¡¡¡¡¡¡¡ HEAD

12 Discussion and Future Work

=======

13 Discussion

¿¿¿¿¿¿¿ master Our motivating question was whether end-to-end low-precision data representation can enable efficient computation with convergence guarantees. We show that a relatively simple stochastic quantization framework can achieve this for linear models. With this setting, as little as two bits per model dimension are sufficient for good accuracy, and can enable a fast FPGA implementation.

¡¡¡¡¡¡¡ HEAD

Non-Linear Models. We mainly discussed linear models (e.g. linear regression) in this paper. The natural question is that can we extend our ZipML framework to non-linear models (e.g. logistic regression or SVM) which are arguably more commonly used? Actually, we find that our framework can be generalized to non-linear settings, and that in practice 8-bit is sufficient to achieve good accuracy on a variety of tasks, such as SVM and logistic regression. However, we notice that the straw man approach, which applies naive stochastic rounding over the input data to just 8-bit precision, converges to similar results, without the added complexity. It is interesting to consider the rationale behind these results. Our framework is based on the idea of *unbiased approximation* of the original SGD process. For linear models, this is easy to achieve. For non-linear models, this is harder, and we focus on guaranteeing arbitrarily low bias. However, for a variety of interesting functions such as hinge loss, guaranteeing low bias requires complex approximations. In turn, these increase the variance. The complexity of the approximation and the resulting variance increase force us to increase the *density* of the quantization, in order to achieve good approximation guarantees. We will leave detailed research for ZipML on non-linear models as future work.

Hardware Selection. We choose to realize our implementation using FPGA because of its flexibility in dealing with low-precision arithmetic, while CPU or GPU can only do at least 8-bit computation efficiently. However, we do observe speed up in other environments – for example, the double sampling techniques are currently being applied to sensor networks with embedded GPUs and CPUs to achieve similar speedup. We are currently conducting an architecture exploration study which aims at understanding the system trade-off between FPGA, CPU, and GPU. This requires us to push the implementation on all three architectures to the extreme. We expect this study will soon provide a full systematic answer to the question that on which hardware can we get the most from the ZipML framework. ======= For non-linear models, the picture is more nuanced. In particular, we find that our framework can be generalized to this setting, and that in practice *8-bit is sufficient* to achieve good accuracy on a variety of tasks, such as SVM and logistic regression. However, in this generalized setting, naive rounding has similar performance on many practical tasks.

It is interesting to consider the rationale behind these results. Our framework is based on the idea of *unbiased approximation* of the original SGD process. For linear models, this is easy to achieve. For non-linear models, this is harder, and we focus on guaranteeing arbitrarily low bias. However, for a variety of interesting functions such as hinge loss, guaranteeing low bias requires complex approximations. In turn, these increase the variance. The complexity of the approximation and the resulting variance increase force us to increase the *density* of the quantization, in order to achieve good approximation guarantees.

Supplemental Materials: Training Models with End-to-End Low Precision: The Cans, the Cannots, and a Little Bit of Deep Learning

This supplemental material is the laboratory of this project. All omitted proofs, additional theorems, and experiment details can be found from corresponding sections.

14 Preliminaries

14.1 Computational Model

We consider a computational model illustrated in Figure **??**. In this context, SGD is often bounded by the bandwidth of data movements cross these components. In particular, we consider the convergence properties of the algorithm when a lossy compression scheme is applied to the data (samples), gradient, and model, for the purpose of reducing the communication cost of the algorithm. It is interesting to consider how lossy compression impacts the update step in SGD. Let $Q(\mathbf{v})$ denote the compression scheme applied to a vector $\mathbf{v}$.

- **Original iteration**:
$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \gamma \mathbf{g}_k(\mathbf{x}_t, \mathbf{a}_t).$$
- **Compressed gradient**:
$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \gamma Q(\mathbf{g}_k(\mathbf{x}_t, \mathbf{a}_t)).$$
- **Compressed model**:
$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \gamma \mathbf{g}_k(Q(\mathbf{x}_t), \mathbf{a}_t).$$
- **Compressed sample**:
$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \gamma \mathbf{g}_k(\mathbf{x}_t, Q(\mathbf{a}_t)).$$
- **End-to-end compression**:
$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \gamma Q(\mathbf{g}_k(Q(\mathbf{x}_t), Q(\mathbf{a}_t))).$$

14.2 Guarantees for SGD

In this paper we consider SGD, a general family of stochastic first order methods for finding the minima of convex (and non-convex) functions. Due to its generality and usefulness, there is a vast literature on SGD in a variety of settings, with different guarantees in all of these settings. Our techniques apply fairly generally in a black box fashion to many of these settings, and so for simplicity we will restrict our attention to a fairly basic setting. For a more comprehensive treatment, see **?**.

Throughout the paper, we will assume the following setting in our theoretical analysis. Let $\mathcal{X} \subseteq \mathbb{R}^n$ be a known convex set, and let $f : \mathcal{X} \to \mathbb{R}$ be differentiable, convex, and unknown. We will assume the following, standard smoothness condition on f:

Definition 1 (Smoothness)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable and convex. We say that it is L-smooth if for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, we have*
$$0 \le f(\mathbf{x}) - f(\mathbf{y}) - \nabla f(\mathbf{y})^T(\mathbf{x}-\mathbf{y}) \le \frac{L}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 .$$

We assume repeated access to stochastic gradients, which on (possibly random) input $\mathbf{x}$, outputs a direction which is in expectation the correct direction to move in. Formally:

Definition 2. *Fix $f : \mathcal{X} \to \mathbb{R}$. A* stochastic gradient *for f with bias bound β is a random function $g(\mathbf{x})$ so that $\mathbb{E}[g(\mathbf{x})] = G(\mathbf{x})$, where $\|G(\mathbf{x}) - \nabla f(\mathbf{x})\|_2 \le \beta$ for all $x \in \mathcal{X}$. We say the stochastic gradient has second moment at most B if $\mathbb{E}[\|g\|_2^2] \le B$ for all $\mathbf{x} \in \mathcal{X}$. We say it has variance at most σ^2 if $\mathbb{E}[\|g(\mathbf{x}) - \nabla f(\mathbf{x})\|_2^2] \le \sigma^2$ for all $\mathbf{x} \in \mathcal{X}$.*

For simplicity, if $\beta = 0$ we will simply refer to such a random function as a stochastic gradient. Under these conditions, the following convergence rate for SGD is well-known:

Theorem 3 (e.g. **?**, Theorem 6.3)**.** *Let $\mathcal{X} \subseteq \mathbb{R}^n$ be convex, and let $f : \mathcal{X} \to \mathbb{R}$ be an unknown, convex, and L-smooth. Let $\mathbf{x}_0 \in \mathcal{X}$ be given, and let $R^2 = \sup_{\mathbf{x}\in\mathcal{X}} \|\mathbf{x}-\mathbf{x}_0\|_2^2$. Suppose we run projected SGD on f with access to independent stochastic gradients with bias bound β and variance bound σ^2 for T steps, with step size $\eta_t = 1/(L+\gamma^{-1})$, where $\gamma = \frac{R}{\sigma}\sqrt{\frac{2}{T}}$, and*

$$T = O\left(R^2 \cdot \max\left(\frac{2\sigma^2}{\epsilon^2}, \frac{L}{\epsilon}\right)\right) . \tag{8}$$

Then $\mathbb{E}\left[f\left(\frac{1}{T}\sum_{t=0}^{T}\mathbf{x}_t\right)\right] - \min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) \le \epsilon + R\beta + \frac{\eta}{2}\beta^2$.

In particular, note that the complexity the SGD method is mainly controlled by the variance bound σ^2 we may obtain. If $\sigma = 0$, the complexity is consistent with the stochastic gradient.

14.3 Randomized Quantization

In this section, we give a procedure to quantize a vector or real values randomly, reducing its information content. We will denote this quantization function by $Q(\mathbf{v}, s)$, where $s \ge 1$ is the tuning parameter. Let $M(\mathbf{v}) : \mathbb{R}^n \to \mathbb{R}^n$ be a positive scaling function such that, for $\mathbf{v} \in \mathbb{R}^n$, $\frac{\mathbf{v}_i}{M_i(\mathbf{v})} \in [-1, 1]$, where $M_i(\mathbf{v})$ denotes the ith element of $M(\mathbf{v})$. For $\mathbf{v} \neq \mathbf{0}$ we define

$$Q_i(\mathbf{v}, s) = M_i(\mathbf{v}) \cdot \mathrm{sgn}(\mathbf{v}_i) \cdot \mu_i(\mathbf{v}, s) , \tag{9}$$

where $\mu_i(\mathbf{v}, s)$'s are independent random variables defined as follows. Let $0 \le \ell < s$ be an integer such that $|\mathbf{v}_i|/M_i(\mathbf{v}) \in [\ell/s, (\ell+1)/s]$, that is, $\ell = \lfloor s|\mathbf{v}_i|/\|\mathbf{v}\|\rfloor$. Here, $p(x, s) = xs - \ell$ for any $x \in [0, 1]$. Then

$$\mu_i(\mathbf{v}, s) = \begin{cases} \ell/s & \text{with probability } 1 - p\left(\frac{|\mathbf{v}_i|}{M(\mathbf{v})}, s\right); \\ (\ell+1)/s & \text{otherwise.} \end{cases}$$

If $\mathbf{v} = \mathbf{0}$, then we define $Q(\mathbf{v}, s) = \mathbf{0}$. For any such choice of M_i, we have the following properties, which generalize Lemma 3.4 in **?**. The proofs follow immediately from those in **?**, and so we omit them for conciseness.

Lemma 6. *For any $\mathbf{v} \in \mathbb{R}^n$, we have that*

- *(Sparsity) $\mathbb{E}[\|Q(\mathbf{v}, s)\|_0] \le s^2 + \sqrt{n}$,*
- *(Unbiasedness) $\mathbb{E}[Q(\mathbf{v}, s)] = \mathbf{v}$, and*
- *(Second Moment Bound) $\mathbb{E}[\|Q(\mathbf{v}, s)\|_2^2] \le rM^2$, where $M = \max_i M_i(\mathbf{v})$, and*

$$r = r(s) = \left(1 + \frac{1}{s^2}\sum_{i=1}^{n} p\left(\frac{|\mathbf{v}_i|}{M_i}, s\right)\right) .$$

We now discuss different choices of the scaling function $M_i(\mathbf{v})$.

"Row Scaling" One obvious choice that was suggested in **?** is to have $M_i(\mathbf{v}) = \|\mathbf{v}\|_2$, in this way, we always have $\frac{\mathbf{v}_i}{M_i(\mathbf{v})} \in [-1, 1]$ and all $M_i(\mathbf{v})$ are the same such that we can store them only once. When the In the following, we will often use the version with $s = 1$, which is as follows.

$$Q_i(\mathbf{v}) = \|\mathbf{v}\|_2 \cdot \mathrm{sgn}(\mathbf{v}_i) \cdot \mu_i(\mathbf{v}) , \tag{10}$$

where $\mu_i(\mathbf{v})$'s are independent random variables such that $\mu_i(\mathbf{v}) = 1$ with probability $|\mathbf{v}_i|/\|\mathbf{v}\|_2$, and $\mu_i(\mathbf{v}) = 0$, otherwise. If $\mathbf{v} = \mathbf{0}$, we define $Q(\mathbf{v}) = \mathbf{0}$. Obviously, if all vectors $\mathbf{v}$ are scaled to have unit ℓ_2 norms, $M(\mathbf{v}) \equiv 1$ and therefore, we can also omit this term. Moreover, it was shown in **?** that for this choice of M_i, the function r can be upper bounded by

$$r(s) \le r_{\mathrm{row}}(s) = 1 + \min\left(\frac{n}{s^2}, \frac{\sqrt{n}}{s}\right) .$$

"Column Scaling" Let $\mathbf{v} \in \mathbb{R}^n$ be a sample and $V \subset \mathbb{R}^n$ be the set of sample vectors. We can obtain the upper and lower bound for each feature, that is,

$$\text{min}_i \leq \mathbf{v}_i \leq \text{max}_i \quad \mathbf{v} \in V$$

is to have $M_i(\mathbf{v}) = \max(|\text{min}_i|, |\text{max}_i|)$. When the input samples are stored as a matrix in which each row corresponds two a vector $\mathbf{v}$, getting $\min_i$ and $\max_i$ is just to getting the $\min$ and $\max$ for each column (feature). Using this scheme, all input samples can share the same $M_i(\mathbf{v})$ and thus can be easily stored in cache when all input samples are accessed sequentially (like in SGD).

Choice between Row Scaling and Column Scaling In this working paper, we make the following choices regarding row scaling and column scaling and leave the more general treatment to future work. For all input samples, we always use column scaling because it is easy to calculate M_i which does not change during training. For all gradients and models, we use row scaling because the range of values is more dynamic.

15 Compressing the Samples for Linear Regression

In this section, we will describe lossy compression schemes for data samples, so that when we apply SGD to solve linear regression on these compressed data points, it still provably converges. Throughout this section, the setting will be as follows. We have labeled data points $(\mathbf{a}_1, b_1), (\mathbf{a}_2, b_2), \ldots, (\mathbf{a}_K, b_K) \in \mathbb{R}^n \times \mathbb{R}$, and our goal is to minimize the function

$$f(\mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K} \|\mathbf{a}_k^\top \mathbf{x} + b_k\|_2^2 \ ,$$

i.e., minimize the empirical least squares loss. The basic (unquantized) SGD scheme which solves this problem is the following: at step $\mathbf{x}_k$, our gradient estimator is $\mathbf{g}'_k = \mathbf{a}_{\pi(k)}(\mathbf{a}_{\pi(k)}^\top \mathbf{x} + b_{\pi(k)})$, where $\pi(k)$ is a uniformly random integer from 1 to m. In a slight abuse of notation we let $\mathbf{a}_k = \mathbf{a}_{\pi(k)}$ for the rest of the section. Then it is not hard to see that $\mathbb{E}[\mathbf{g}'_k] = \nabla f(\mathbf{x}_k)$, and so this is indeed a stochastic gradient.

The rest of the section is now devoted to devising quantization schemes for $\mathbf{g}'_k$ when given access only to $\mathbf{a}_k$ and b_k, namely, given access only to the data points.

15.1 Naive Random Quantization is Biased

As a first exercise, we look at what happens when we work with the data directly in quantized form in the context of linear regression. The gradient becomes

$$\mathbf{g}_k := Q(\mathbf{a}_k, s)Q(\mathbf{a}_k, s)^\top \mathbf{x} + Q(\mathbf{a}_k, s)b_k.$$

It is not hard to see that the expected value of this is in fact:

$$\mathbb{E}[\mathbf{g}_k] := \mathbf{a}_k\mathbf{a}_k^\top \mathbf{x} + \mathbf{a}_k b_k + D_{s,\mathbf{a}}\mathbf{x},$$

where $D_{s,\mathbf{a}}$ is a diagonal matrix and its ith diagonal element is

$$\mathbb{E}[Q(\mathbf{a}_i, s)^2] - \mathbf{a}_i^2.$$

Since $D_{s,\mathbf{a}}$ is non-zero, we obtain a *biased* estimator of the gradient, so the iteration is unlikely to converge. In fact, it is easy to see that in instances where the minimizer $\mathbf{x}$ is large and gradients become small, we will simply diverge. Fortunately, however, this issue can be easily fixed.

15.2 Double Sampling

Algorithm Instead of the naive estimate, our algorithm is as follows. We generate two independent random quantizations Q_1 and Q_2 and revise the gradient as follows:

$$\mathbf{g}_k := Q_1(\mathbf{a}_k, s)(Q_2(\mathbf{a}_k, s)^\top \mathbf{x} + b_k) \ .$$

It is not hard to see that the above is an unbiased estimator of the true gradient.[2]

Variance Analysis

Lemma 7. *The stochastic gradient variance using double sampling above $\mathbb{E}\|\mathbf{g}_t - \mathbf{g}_t^{(full)}\|^2$ can be bounded by*

$$\Theta\left(\mathcal{TV}(\mathbf{a}_t)(\mathcal{TV}(\mathbf{a}_t)\|\mathbf{x}\odot\mathbf{x}\| + \|\mathbf{a}_t^\top\mathbf{x}\|^2 + \|\mathbf{x}\odot\mathbf{x}\|\|\mathbf{a}_t\|^2)\right),$$

where $\mathcal{TV}(\mathbf{a}_t) := \mathbb{E}\|Q(\mathbf{a}_t) - \mathbf{a}_t\|^2$ and $\odot$ denotes the element product.

Proof. Let $\mathbf{a}$ denote $\mathbf{a}_t$ for short in the followed proof.

$$\begin{aligned}
& \mathbb{E}\left\|Q_1(\mathbf{a})(Q_2(\mathbf{a})^\top\mathbf{x} + b_t)\right\|^2 \\
\leq\ & 2\mathbb{E}\left\|(Q_1(\mathbf{a}) - \mathbf{a})Q_2(\mathbf{a})^\top\mathbf{x}\right\|^2 + 2\mathbb{E}\left\|\mathbf{a}(Q_2(\mathbf{a}) - \mathbf{a})^\top\mathbf{x})\right\|^2 \\
\leq\ & 2\mathbb{E}_1\|Q_1(\mathbf{a}) - \mathbf{a}\|^2\mathbb{E}_2(Q_2(\mathbf{a})^\top\mathbf{x})^2 + 2\|\mathbf{a}\|^2\mathbb{E}((Q_2(\mathbf{a}) - \mathbf{a})^\top\mathbf{x})^2 \\
\leq\ & 2\mathbb{E}_1\|Q_1(\mathbf{a}) - \mathbf{a}\|^2\mathbb{E}_2(Q_2(\mathbf{a})^\top\mathbf{x})^2 + 2\|\mathbf{a}\|^2\mathbb{E}((Q_2(\mathbf{a}) - \mathbf{a})^\top\mathbf{x})^2 \\
\leq\ & 2\mathcal{TV}(\mathbf{a})(2\|\mathbf{a}\|^2\mathbb{E}((Q_2(\mathbf{a}) - \mathbf{a})^\top\mathbf{x})^2 + 2(\mathbf{a}^\top\mathbf{x})^2) + 2\|\mathbf{a}\|^2\mathbb{E}((Q_2(\mathbf{a}) - \mathbf{a})^\top\mathbf{x})^2 \\
\leq\ & \Theta\left(\mathcal{TV}(\mathbf{a})(\mathcal{TV}(\mathbf{a})\|\mathbf{x}\odot\mathbf{x}\| + \|\mathbf{a}^\top\mathbf{x}\|^2 + \|\mathbf{x}\odot\mathbf{x}\|\|\mathbf{a}\|^2)\right),
\end{aligned}$$

which completing the proof. □

Let $r = r(s) = 1 + \min(n/s^2, \sqrt{n}/s)$ be the blow-up in the second moment promised in Lemma 6. Then, we have the following lemma.

Lemma 8. *Let $\mathbf{a}_k, \mathbf{x}, b_k$ be fixed, and suppose that $\|\mathbf{a}_k\|_2^2 \leq A^2$, $\|\mathbf{x}\|_2^2 \leq R^2$, and $\max_i M_i(\mathbf{a}_k) \leq M_a$. Let $\mathbf{g}'_k = \mathbf{a}_k(\mathbf{a}_k^\top\mathbf{x} + b)$ be the (unquantized) stochastic gradient update. Then, we have*

$$E_{Q_1,Q_2}[\|\mathbf{g}_k\|_2^2] \leq r\cdot\left(\|\mathbf{g}'_k\|_2^2\cdot\frac{M_a^2}{\|\mathbf{a}_k\|_2^2} + \|\mathbf{a}_k\|_2^2\frac{M_a^2}{s^2}R^2\right) .$$

Proof. We have that

$$\mathbb{E}_{Q_1,Q_2}(\|\mathbf{g}_k\|^2) = \mathbb{E}_{Q_1,Q_2}[\|Q_1(\mathbf{a}_k, s)(Q_2(\mathbf{a}_k, s)^\top\mathbf{x} + b_k)\|_2^2].$$

Next we have

$$\begin{aligned}
\mathbb{E}_{Q_1,Q_2}[\|Q_1(\mathbf{a}_k, s)(Q_2(\mathbf{a}_k, s)^\top\mathbf{x} + b_k)\|_2^2] &= \mathbb{E}_{Q_2}\left[\mathbb{E}_{Q_1}[(Q_2(\mathbf{a}_k, s)^\top\mathbf{x} + b_k)^2 Q_1(\mathbf{a}_k, s)^\top Q_1(\mathbf{a}_k, s)]\right] \\
&= \mathbb{E}_{Q_1}[\|Q_1(\mathbf{a}_k, s)\|_2^2]\cdot\mathbb{E}_{Q_2}[\|\mathbf{a}_k(Q_2(\mathbf{a}_k, s)^\top\mathbf{x} + b_k)\|_2^2] \\
&\leq^{\text{Lemma 6}} rM_a^2\cdot\mathbb{E}[(Q_2(\mathbf{a}_k, s)^\top\mathbf{x} + b_k)^2] \\
&= rM_a^2\left(\mathbb{E}[(Q_2(\mathbf{a}_k, s)^\top\mathbf{x})^2] + 2b_k\mathbb{E}[Q_2(\mathbf{a}_k, s)^\top\mathbf{x}] + b_k^2\right) \\
&= rM_a^2\left(\mathbb{E}[(Q_2(\mathbf{a}_k, s)^\top\mathbf{x})^2] + 2b_k\mathbf{a}_k^\top\mathbf{x} + b_k^2\right)
\end{aligned}$$

[2] In our implementation, we used the average gradient $\mathbf{g}_k := \frac{1}{2}\left(Q_1(\mathbf{a}_k, s)(Q_2(\mathbf{a}_k, s)^\top\mathbf{x} + b_k) + Q_2(\mathbf{a}_k, s)(Q_1(\mathbf{a}_k, s)^\top\mathbf{x} + b_k)\right)$. This version does not impact the upper bound in our variance analysis, but enjoys lower variance (by a constant) both theoretically and empirically.

Moreover, we have

$$\begin{aligned}E[(Q_2(\mathbf{a}_k,s)^\top \mathbf{x})^2] &= \mathbf{x}^\top \left(\mathbb{E}\left[Q_2(\mathbf{a}_k,s)Q_2(\mathbf{a}_k,s)^\top\right]\right)\mathbf{x} \\ &= \mathbf{x}^\top(\mathbf{a}_k\mathbf{a}_k^\top + D)\mathbf{x}^\top \\ &\leq (\mathbf{a}_k^\top \mathbf{x})^2 + \|D\|_{\mathrm{op}}\|\mathbf{x}\|_2^2 ,\end{aligned}$$

where $D = \mathrm{diag}_i[(\mathbb{E}[Q_2(\mathbf{a}_k,s)_i^2]) - (\mathbf{a}_k)_i^2] = \mathrm{diag}_i[\mathrm{Var}[Q_2(\mathbf{a}_k,s)_i]]$. Further, we have that $\|D\|_{\mathrm{op}} \leq M_a^2/s^2$. Therefore we have that:

$$\begin{aligned}\mathbb{E}_{Q_1,Q_2}[\|Q_1(\mathbf{a}_k,s)(Q_2(\mathbf{a}_k,s)^\top\mathbf{x}+b_k)\|_2^2] &\leq rM_a^2\left((\mathbf{a}_k^\top\mathbf{x})^2 + \frac{M_a^2}{s^2}R^2 + 2b_k\mathbf{a}_k^\top\mathbf{x} + b_k^2\right) \\ &= r\left(\|\mathbf{g}_k'\|_2^2 \cdot \frac{M_a^2}{\|\mathbf{a}_k\|_2^2} + \frac{A^2M_a^2R^2}{s^2}\right)\end{aligned}$$

as claimed, since $\|\mathbf{g}_k'\|_2^2 = \|\mathbf{a}_k\|_2^2(\mathbf{a}_k^T\mathbf{x}+b_k)^2$. □

In particular, this implies the following variance bound on our quantized updates:

Corollary 1. *Let $\mathbf{a}_k, \mathbf{x}, b_k, \mathbf{g}_k'$ be as above. Suppose moreover $\mathbb{E}[\|\mathbf{g}_k' - \nabla f(\mathbf{x}_k)\|_2^2] \leq \sigma^2$ and $\mathbb{E}[\|\mathbf{g}_k'\|_2^2] \leq B$. Then, we have*

$$\mathbb{E}\left[\|\mathbf{g}_k - \nabla f(\mathbf{x}_k)\|_2^2\right] \leq \sigma^2 + \left(r\frac{M_a^2}{\|\mathbf{a}_k\|_2^2} - 1\right)B + \frac{rA^2M_a^2R^2}{s^2} ,$$

where the expectation is taken over $\mathbf{g}_k'$ and the randomness of the quantization.

Proof. Observe that $\|\mathbf{g}_k - \nabla f(\mathbf{x}_k)\|_2^2 = \|\mathbf{g}_k - \mathbf{g}_k'\|_2^2 + 2(\mathbf{g}_k - \mathbf{g}_k')^\top(\mathbf{g}_k' - \nabla f(\mathbf{x}_k)) + \|g_k' + \nabla f(\mathbf{x}_k)\|_2^2$. Since $\mathbb{E}[(\mathbf{g}_k - \mathbf{g}_k')^\top(\mathbf{g}_k' - \nabla f(\mathbf{x}_k))] = 0$, and by assumption $\mathbb{E}[\|g_k' + \nabla f(\mathbf{x}_k)\|_2^2] \leq \sigma^2$, it suffices to bound the expectation of the first term. We have

$$\mathbb{E}\left[\|\mathbf{g}_k - \nabla f(\mathbf{x}_k)\|_2^2\right] \leq 2\sigma^2 + 2\mathbb{E}_{\mathbf{g}_k'}\left[\mathbb{E}_{Q_1,Q_2}[\|\mathbf{g}_k' - \mathbf{g}_k\|_2^2 \mid \mathbf{g}_k']\right] .$$

Since $\mathbb{E}_{Q_1,Q_2}[\mathbf{g}_k|\mathbf{g}_k'] = \mathbf{g}_k'$, we have that

$$\begin{aligned}\mathbb{E}_{Q_1,Q_2}[\|\mathbf{g}_k' - \mathbf{g}_k\|_2^2 \mid \mathbf{g}_k'] &= \mathbb{E}_{Q_1,Q_2}[\|\mathbf{g}_k\|_2^2|\mathbf{g}_k'] - \|\mathbf{g}_k'\|_2^2 \\ &\leq \left(r\frac{M_a^2}{\|\mathbf{a}_k\|_2^2} - 1\right)\|\mathbf{g}_k'\|_2^2 + \frac{rA^2M_a^2R^2}{s^2} ,\end{aligned}$$

from which the corollary follows. □

In particular, observe that this corollary essentially suggests that the quantized stochastic gradient variance is bounded by

$$\mathbb{E}\left[\|\mathbf{g}_k - \nabla f(\mathbf{x}_k)\|_2^2\right] \leq \sigma^2 + \Theta(n/s^2)$$

in the scenario when $M_i(\mathbf{v}) = \|\mathbf{v}\|_2$. The first term σ^2 is due to using stochastic gradient, while the second term is caused by quantization. The value of s is equal to $\lceil(2^b - 1)/2\rceil$. Therefore, to ensure these two terms are comparable (so as not to degrade the convergence time of quantized stochastic gradient), the number of bits needs to be greater than $\Theta(\log n/\sigma)$.

16 Quantizing the Model

We now assume the setting where the processor can only work with the model in *quantized* form when computing the gradients. However, the gradient is stored in full precision—the model is quantized only when communicated. The gradient computation in this case is:

$$\mathbf{g}_k := \mathbf{a}_k\mathbf{a}_k^\top Q(\mathbf{x},s) + \mathbf{a}_k b_k. \tag{11}$$

It is easy to see that this gradient is unbiased, as the quantizer commutes with the (linear) gradient.

$$\mathbb{E}[\mathbf{g}_k] := \mathbf{a}_k\mathbf{a}_k^\top \mathbb{E}[Q(\mathbf{x}, s)] + \mathbf{a}_k b_k = \mathbf{a}_k\mathbf{a}_k^\top \mathbf{x} + \mathbf{a}_k b_k = \mathbf{g}_k.$$

Further, the second moment bound is only increased by the variance of the quantization.

Lemma 9. *Let $\mathbf{a}_k, \mathbf{x}, b_k$ be fixed, and suppose that $\|\mathbf{a}_k\|_2^2 \leq A^2$, and $\max_i M_i(\mathbf{x}) \leq M_x$. Let $\mathbf{g}'_k = \mathbf{a}_k(\mathbf{a}_k^\top \mathbf{x} + b_k)$ be the (unquantized) stochastic gradient update. Then, we have*

$$\mathbb{E}[\|\mathbf{g}_k\|_2^2] \leq \|\mathbf{g}'_k\|_2^2 + \frac{A^4 M_x^2}{s^2} \ .$$

Proof. We have

$$\begin{aligned}\mathbb{E}[\|\mathbf{g}_k\|_2^2] &= \|\mathbf{a}_k\|_2^2 \mathbb{E}\left[\left(\mathbf{a}_k^\top Q(\mathbf{x}, s) + b_k\right)^2\right] \\ &= \|\mathbf{a}_k\|_2^2 \left(a_k^\top \mathbb{E}[Q(\mathbf{x}, s)Q(\mathbf{x}, s)^\top] a_k + 2b_k \mathbb{E}[Q(\mathbf{x}, s)^\top \mathbf{a}_k] + b_k^2\right) \\ &= \|\mathbf{a}_k\|_2^2 \left(a_k^\top \mathbb{E}[Q(\mathbf{x}, s)Q(\mathbf{x}, s)^\top] a_k + 2b_k \mathbf{x}^\top \mathbf{a}_k + b_k^2\right) \ .\end{aligned}$$

As we had previously for double sampling, we have

$$\begin{aligned}\mathbf{a}_k^\top \left(\mathbb{E}\left[Q_2(\mathbf{x}, s)Q_2(\mathbf{x}, s)^\top\right]\right) \mathbf{a}_k &= \mathbf{a}_k^\top (\mathbf{x}\mathbf{x}^\top + D)\mathbf{a}_k^\top \\ &\leq (\mathbf{a}_k^\top \mathbf{x})^2 + \|D\|_{\mathrm{op}} \|\mathbf{a}_k\|_2^2 \ ,\end{aligned}$$

where as before we have that D consists of diagonal elements $\mathbb{E}[Q_2(\mathbf{x}, s)_i^2]) - (\mathbf{x})_i^2 = [\mathrm{Var}[Q_2(\mathbf{x}, s)_i]] \leq M_x^2/s^2$. Hence altogether we have

$$\mathbb{E}[\|\mathbf{g}_k\|_2^2] \leq \|\mathbf{g}'_k\|_2^2 + \frac{A^4 M_x^2}{s^2} \ ,$$

as claimed. □

17 Quantizing the Gradients

Recent work has focused on quantizing the gradients with low-precision representation. We omit the description of this direction because it is relatively well-studied and is orthogonal to the contribution of this paper. From Lemma 6, we have:

Lemma 10. *Gradient quantization increases the second moment bound of the gradient by a multiplicative rM^2 factor.*

18 End-to-end Quantization

We describe the end-to-end quantization strategy of quantizing gradients, model, and input samples all at the same time. We assume all 3 sources are quantized: Gradient, model and data. However, the update to the model happens in full precision. The gradient becomes:

$$\mathbf{g}_k := Q_4\left(Q_1(\mathbf{a}_k, s)(Q_2(\mathbf{a}_k, s)^\top Q_3(\mathbf{x}, s) + b_k), s\right). \tag{12}$$

Here $Q_1, \ldots, Q_4$ are all independent quantizations. Q_3 and Q_4 are normalized with row scaling, and Q_1, Q_2 can be normalized arbitrarily. The iteration then is:

$$\mathbf{x} = \mathbf{x} - \gamma \mathbf{g}_k. \tag{13}$$

From combining the previous results, we obtain that, if the samples are normalized, the following holds:

Corollary 2. *Let $\mathbf{a}_k, \mathbf{x}, b_k$ be so that $\|\mathbf{a}_k\|_2^2 \leq 1, \|\mathbf{x}\|_2^2 \leq R^2$. Let M_a, M_x be as above, and let $\mathbf{g}'_k = \mathbf{a}_k(\mathbf{a}_k^\top \mathbf{x} + b_k)$ be the (unquantized) stochastic gradient. Then, we have*

$$\mathbb{E}[\|\mathbf{g}_k\|_2^2] \leq r_{\text{row}} \cdot \left(r M_a \left(\|\mathbf{g}'_k\|_2^2 + \frac{R^2}{s^2} \right) + \frac{r^2 M_a^2 R^2}{s^2} \right) .$$

By a calculation identical to the proof of Cor 1, we obtain:

Corollary 3. *Let $\mathbf{a}_k, \mathbf{x}, b_k$ be so that $\|\mathbf{a}_k\|_2^2 \leq 1, \|\mathbf{x}\|_2^2 \leq R^2$. Let M_a, M_x be as above, and let $\mathbf{g}'_k = \mathbf{a}_k(\mathbf{a}_k^\top \mathbf{x} + b_k)$ be the (unquantized) stochastic gradient. Then, we have*

$$\mathbb{E}[\|\mathbf{g}_k - \nabla f(\mathbf{x}_k)\|_2^2] \leq \sigma^2 + r_{\text{row}} \cdot \left(r M_a \left(\|\mathbf{g}'_k\|_2^2 + \frac{R^2}{s^2} \right) + \frac{r^2 M_a^2 R^2}{s^2} \right) .$$

Plugging this into Theorem 3 gives the bounds for convergence of these end-to-end quantization methods with SGD.

19 Extension to Classification Models

19.1 Least Squares Support Vector Machines

We first extend our quantization framework to least squares support vector machines–a model popularly used for classification tasks and often showing comparable accuracy to SVM **?**. The Least Squares SVM optimization problem is formally defined as follows:

$$\min_{\mathbf{x}} : \quad \frac{1}{2K} \sum_{k=1}^{K} (1 - b_k \mathbf{a}_k^\top \mathbf{x})^2 + \frac{c}{2} \|\mathbf{x}\|^2$$

Without loss of generality, we assume two-class classification problems, i.e. $b_k \in \{-1, 1\}$. We now have:

$$\min_{\mathbf{x}} : \quad \frac{1}{2K} \sum_{k=1}^{K} (\mathbf{a}_k^\top \mathbf{x} - b_k)^2 + \frac{c}{2} \|\mathbf{x}\|^2$$

where c is the regularization parameter. The gradient at a randomly selected sample$(\mathbf{a}_k, b_k)$ is:

$$\mathbf{g}'_k := \mathbf{a}_k \mathbf{a}_k^\top \mathbf{x} + \mathbf{a}_k b_k + \frac{c}{k} \mathbf{x}.$$

The gradient is similar to regularized linear regression (Eq. 11). In particular, the only difference is the additional $\mathbf{x}$ term. Since we can quantize this term separately using an additional quantization, and we can quantize first term using the techniques above, we can still use the same quantization framework we developed for linear regression.

20 Support Vector Machines

Consider solving the following hinge loss optimization problem for Support Vector Machines(SVM):

$$\min_{\|\mathbf{x}\|_2 \leq R} : \quad \sum_{k=1}^{K} \max(0, 1 - b_k \mathbf{a}_k^\top \mathbf{x}) .$$

The (sub-)gradient at a randomly selected sample $(\mathbf{a}_k, b_k)$ is:

$$\mathbf{g}'_k := \begin{cases} -b_k \mathbf{a}_k & \text{if } b_k \mathbf{a}_k^\top \mathbf{x} < 1; \\ 0 & \text{otherwise.} \end{cases}$$

Observe that this loss function is not smooth.[3] When quantizing samples, the estimator of gradient is biased, as $(1 - b_k \mathbf{a}_k^\top \mathbf{x})$ and $(1 - b_k Q(\mathbf{a}_k)^\top \mathbf{x})$ may have different signs, in which case the two procedures will apply different gradients. We say that in this case the gradient is *flipped*. We have two approaches to dealing with this: the first provides rigorous guarantees, however, requires some fairly heavy algorithmic machinery (in particular, Johnson-Lindenstrauss matrices with little randomness). The latter is a simpler heuristic that we find works well in practice.

20.1 Polynomial approximation and ℓ_2-refetching via Johnson-Lindenstrauss

Let $H(x)$ be the Heaviside function, i.e. $H(x) = 1$ if $x \geq 0$ and $H(x) = 0$ if $x < 0$. For some fixed parameters ϵ, δ, we take a degree d polynomial P so that $|P(x) - H(x)| \leq \epsilon$ for all $x \in [-(R^2+1), R^2+1] \setminus [-\delta, \delta]$, and so that $|P(x)| \leq 1$ for all $x \in [-(R^2+1), R^2+1]$. Since the gradient of the SVM loss may be written as $\mathbf{g}'_k = -H(1 - b_k \mathbf{a}_k^\top \mathbf{x}) b_k \mathbf{a}_k$, we will let Q be a random quantization of $P(1 - b_k \mathbf{a}_k^\top \mathbf{x})$ (as described in the main paper), and our quantized gradient will be written as $\mathbf{g}_k = -Q(1 - b_k \mathbf{a}_k^\top \mathbf{x}) b_k Q_2(\mathbf{a}_k)$, where Q_2 is an independent quantization of $\mathbf{a}_k$. We also define

$$r(s) = \max_{\mathbf{a}_k} \mathbb{E}[\|\mathbf{g}_k\|_2^2]$$

to be a bound on the second moment of our $\mathbf{g}_k$, for any random choice of $\mathbf{a}_k$.

However, the polynomial approximation offers no guarantees when $1 - b_k \mathbf{a}_k^\top \mathbf{x} \in [-\delta, \delta]$, and thus this provides no black box guarantees on error convergence. We have two approaches to avoid this problem. Our first result shows that under reasonable generative conditions, SGD without additional tricks still provides somewhat non-trivial guarantees. However, in general it cannot provide guarantees up to error ϵ, as one would naively hope. We then describe a technique which always allows us to obtain error ϵ, however, requires refetching. We show that under the same generative conditions, we do not need to refetch very often.

Throughout this subsection, we will assume that the a spectral norm bound on the second moment of the data points, we should not refetch often. Such an assumption is quite natural: it should happen for instance if (before rescaling) the data comes from any distribution whose covariance has bounded second moment. Formally:

Definition 3. *A set of data points $\mathbf{a}_1, \ldots, \mathbf{a}_m$ is C-*isotropic *if $\| \sum_{i=1}^m \mathbf{a}_i \mathbf{a}_i^\top \| \leq C$, where $\| \cdot \|$ denotes the operator norm of the matrix.*

20.2 SGD for C-isotropic data

Our first result is the following:

Theorem 4. *Suppose the data $\mathbf{a}_i$ is C-isotropic, and $\|\mathbf{a}_i\|_2 \leq 1$ for all i. Suppose $\mathbf{g}'_k$ is an unbiased stochastic gradient for f with variance bound σ^2. Then $\mathbf{g}_k$ is a $\epsilon + \frac{R}{mC(1-\delta)^2}$ biased stochastic gradient for $\nabla f(\mathbf{x})$ with variance bound $\sigma^2 + r(s) + \epsilon^2 + (r(s)+4)\frac{R}{mC(1-\delta)^2}$.*

In particular, this implies that if $\frac{R}{mC(1-\delta)^2} = O(\epsilon)$, this bias does not asymptotically change our error, and the variance bound increases by as much as we would expect without the biased-ness of the gradient. Before we prove Theorem 4, we need the following lemma:

Lemma 11. *Suppose $\mathbf{a}_1, \ldots, \mathbf{a}_m$ are C-isotropic, and let $\|\mathbf{x}\|_2 \leq R$. Then, the number of points L satisfying $1 - b_k \mathbf{a}_k \mathbf{x} \in [-\delta, \delta]$ satisfies $L \leq \frac{R}{C(1-\delta)^2}$.*

Proof. Observe that any such point satisfies $(\mathbf{a}_k^\top \mathbf{x})^2 \geq (1-\delta)^2$. Then, by the spectral norm assumption, we have $C\|\mathbf{x}\|_2^2 \geq \sum_{i=1}^m (\mathbf{a}_i^\top \mathbf{x})^2 \geq L(1-\delta)^2$, which implies that $L \leq \frac{R}{C(1-\delta)^2}$. □

Proof of Theorem 4. We first prove the bias bound. Let $\mathcal{S}$ be the set of points $\mathbf{a}_k$ so that $1 - b_k \mathbf{a}_k \mathbf{x} \in [-\delta, \delta]$. By the above, we have that $|\mathcal{S}| \leq \frac{R}{C(1-\delta)^2}$. Moreover, if $\mathbf{a}_k \notin \mathcal{S}$, we have by assumption

$$\begin{aligned}\|\mathbb{E}_{\mathbf{g}_k}[\mathbf{g}_k|\mathbf{a}_k] - \mathbf{g}'_k\| &= |P(1 - b_k \mathbf{a}_k \mathbf{x}) - H(1 - b_k \mathbf{a}_k \mathbf{x})| \|\mathbf{a}_k\|_2 \\ &\leq \epsilon \;.\end{aligned}$$

[3] Technically this implies that Theorem 3 does not apply in this setting, but other well-known and similar results still do, see **?**.

Moreover, for any $\mathbf{a}_k$, we always have $\|\mathbb{E}_{\mathbf{g}_k}[\mathbf{g}_k|\mathbf{a}_k]\|_2 \leq \mathbb{E}_{\mathbf{g}_k}[\|\mathbf{g}_k\||\mathbf{a}_k] \leq 1$. Therefore, we have

$$\begin{aligned}
\|\mathbb{E}_{\mathbf{a}_k}\mathbb{E}_{\mathbf{g}_k}[\mathbf{g}_k] - \nabla f(\mathbf{x})\| &= \|\mathbb{E}_{\mathbf{a}_k}\mathbb{E}_{\mathbf{g}_k}[\mathbf{g}_k - \mathbf{g}'_k]\|_2 \\
&\leq \frac{1}{m}\left(\sum_{\mathbf{a}_k \notin \mathcal{S}} \|\mathbb{E}_{\mathbf{g}_k}[\mathbf{g}_k - \mathbf{g}'_k|\mathbf{a}_k]\|_2 + \sum_{\mathbf{a}_k \in \mathcal{S}} \|\mathbb{E}_{\mathbf{g}_k}[\mathbf{g}_k - \mathbf{g}'_k||\mathbf{a}_k]\|_2\right) \\
&\leq \frac{1}{m}\left(\epsilon|\mathcal{S}^c| + |\mathcal{S}|\right) \\
&\leq \epsilon + \frac{R}{mC(1-\delta)^2} \ .
\end{aligned}$$

We now prove the variance bound. Observe that if $\mathbf{a}_k \notin \mathcal{S}$, then

$$\begin{aligned}
\mathbb{E}[\|\mathbf{g}_k - \mathbf{g}'_k\|_2^2|\mathbf{a}_k] &= \mathbb{E}[\|\mathbf{g}_k - \mathbb{E}[\mathbf{g}_k|\mathbf{a}_k]\|_2^2|\mathbf{a}_k] + \|\mathbb{E}[\mathbf{g}_k|\mathbf{a}_k] - \mathbf{g}'_k\|_2^2 \\
&\leq r(s) + \epsilon^2 \ .
\end{aligned}$$

On the other hand, if $\mathbf{a}_k \in \mathcal{S}$, then by the inequality $(a+b)^2 \leq 2a^2 + b^2$ we still have the weaker bound

$$\begin{aligned}
\mathbb{E}[\|\mathbf{g}_k - \mathbf{g}'_k\|_2^2|\mathbf{a}_k] &= \mathbb{E}[\|\mathbf{g}_k - \mathbb{E}[\mathbf{g}_k|\mathbf{a}_k]\|_2^2|\mathbf{a}_k] + \|\mathbb{E}[\mathbf{g}_k|\mathbf{a}_k] - \mathbf{g}'_k\|_2^2 \\
&\leq r(s) + 2\mathbb{E}[\|\mathbf{g}_k\|_2^2|\mathbf{a}_k] + 2\|\mathbf{g}'_k\|_2^2 \\
&\leq r(s) + 4 \ ,
\end{aligned}$$

since $\|\mathbf{g}_k\|_2^2 \leq \|\mathbf{a}_k\|_2^2 \leq 1$ and similarly for $\mathbf{g}'_k$. Thus, we have

$$\begin{aligned}
\mathbb{E}[\|\mathbf{g}_k - \nabla f(\mathbf{x})\|_2^2] &= \sigma^2 + \mathbb{E}[\|\mathbf{g}_k - \mathbf{g}'_k\|_2^2] \\
&= \sigma^2 + \frac{1}{m}\left(\sum_{\mathbf{a}_k \notin \mathcal{S}} \|\mathbb{E}_{\mathbf{g}_k}[\|\mathbf{g}_k - \mathbf{g}'_k\|_2^2|\mathbf{a}_k]\|_2 + \sum_{\mathbf{a}_k \in \mathcal{S}} \|\mathbb{E}_{\mathbf{g}_k}[\|\mathbf{g}_k - \mathbf{g}'_k\|_2^2||\mathbf{a}_k \in \mathcal{S}]\|_2\right) \\
&\leq \sigma^2 + \frac{1}{m}\left((r(s) + \epsilon^2)\cdot|\mathcal{S}^c| + (r(s)+4)\cdot|\mathcal{S}|\right) \\
&\leq \sigma^2 + r(s) + \epsilon^2 + (r(s)+4)\frac{R}{mC(1-\delta)^2} \ ,
\end{aligned}$$

as claimed. □

20.3 ℓ_2-refetching

One drawback of the approach outlined above is that in general, if $\frac{R}{mC(1-\delta)^2}$ is large, then this method does not provide any provable guarantees. In this section, we show that it is still possible, with some additional preprocessing, to provide non-trivial guarantees in this setting, without increasing the communication that much.

Our approach will be to estimate this quantity using little communication per round, and then refetch the data points if $1 - b_k\mathbf{a}_k^\top\mathbf{x} \in [-\delta, \delta]$. We show that under reasonable generative assumptions on the $\mathbf{a}_k$, we will not have to refetch very often.

20.3.1 ℓ_2-refetching using Johnson-Lindenstrauss

For scalars a, b and $\gamma \in [0,1)$, we will use $a \leq_\gamma b$ to mean $a \leq e^\gamma b$, and $a \approx_\gamma b$ to denote that $e^{-\gamma}a \leq b \leq e^\gamma a$.

We require the following theorem:

Theorem 5. *Fix $\gamma, \tau > 0$, n. Then, there is a distribution $\mathcal{D}$ over $n \times r$ matrices which take values in ± 1 so that if M is drawn from $\mathcal{D}$, then for any $x \in \mathbb{R}^n$, we have $\|x\|_2 \approx_\gamma \|Mx\|_2$ with probability $1-\tau$. If the processors have shared randomness, the distribution can be sampled in time $O(nr)$.*

Otherwise, the distribution can be sampled from in time $O(n\log n + \mathrm{poly}(r))$, and using only

$$\alpha(n,\gamma,\tau) := O\left(\log n + \log(1/\tau)\cdot\log\left(\frac{\log 1/\tau}{\gamma}\right)\right)$$

bits of randomness.

If $M \sim \mathcal{D}$, we will call M a *JL matrix*. In the remainder, we will assume that the processors have shared randomness, for instance by using a pseudo-random number generator initialized with the same seed. We will use this shared randomness solely to sample the same M between the two processors. Otherwise, one processor may simply send $\alpha(n,\gamma,\tau)$ random bits to the other, and it is easily verified this does not change the asymptotic amount of communication required.

As a corollary of this, we have:

Corollary 4. *Fix δ,τ. Suppose one processor has $\mathbf{a}_k$ and another has $\mathbf{x}$. Suppose furthermore that $\|\mathbf{a}_k\|_2^2 \le 1$, and $\|\mathbf{x}\|_2^2 \le R^2$, where $R \ge 1$. There is a protocol which with probability $1-\tau$ outputs a c so that $|c-(1-b_k\mathbf{a}_k^\top\mathbf{x}_k)| \le \delta$ that requires each processor to send*

$$O\left(R^2\frac{\log(1/\tau)\log(n/\delta)}{\gamma^2}\right) \textit{ bits.}$$

Proof. The protocol is as follows. Let $\gamma' = O(\gamma/R)$, and let r be as above, except with γ'. Using these shared random bits, have both processors sample the same $M \sim \mathcal{D}$. Then, have the processors send $M\mathbf{a}_k$ and $M\mathbf{x}$ up to $O(\log n/\delta)$ bits of precision per coordinate. Using these vectors, compute the quantities $\|M(\mathbf{a}_k-\mathbf{x})\|_2^2, \|M\mathbf{a}_k\|_2^2, \|M\mathbf{x}\|_2^2$ up to additive error $O(\delta)$. Then, output $c = 1 - b_k(\|M\mathbf{a}_k - M\mathbf{x}\|_2^2 - (\|M\mathbf{a}_k\|_2^2 + \|M\mathbf{x}\|_2^2))$.

That this protocol sends the correct number of bits follows from the description of the algorithm. Thus it suffices to prove correctness. By Theorem 5 and a union bound, we have that $\|M(\mathbf{a}_k-\mathbf{x})\|_2^2 \approx_{2\gamma} \|\mathbf{a}_k-\mathbf{x}\|_2^2$, $\|M\mathbf{a}_k\|_2^2 \approx_{2\gamma} \|\mathbf{a}_k\|_2^2$, $\|M\mathbf{x}\|_2^2 \approx_{2\gamma} \|\mathbf{x}\|_2^2$ with probability $1-\tau$. Let us condition on this event for the rest of the proof. We have $\|x\|_2^2 \le R^2$ and so by a triangle inequality, $\|\mathbf{a}_k - x\|_2^2 \le (\sqrt{R}+1)^2$. Thus, by our choice of γ, we have that $|\|Mv\|_2^2 - \|v\|_2^2| \le O(\delta)$, for all $v \in \{\mathbf{a}_k - x, \mathbf{a}_k, \mathbf{x}\}$. Thus, since $2\mathbf{a}_k^\top\mathbf{x} = \|\mathbf{a}_k-\mathbf{x}\|_2^2 - (\|\mathbf{a}_k\|_2^2 + \|\mathbf{x}\|_2^2)$, this implies that by an appropriate setting of parameters, we have that $|c-(1-b_k\mathbf{a}_k^\top\mathbf{x}_k)| \le \delta$, as claimed. □

Thus, our algorithm for computing the gradient for SVM is as follows.

- Use the protocol from Corollary 4 with $\tau = O(\delta)$ to compute a c so that with probability $1-\delta$ we have $|c-(1-b_k\mathbf{a}_k^\top\mathbf{x})| \le \delta$.
- If $|c| \le 2\delta$, we refetch and compute the full (unquantized) gradient $\mathbf{g}'_k$.
- Otherwise, we output the polynomial approximation $\mathbf{g}_k$.

Then, our result is the following:

Theorem 6. *Let $\hat{\mathbf{g}}_k$ be the estimator produced by the above procedure. Assume that $\|\mathbf{a}_k\|_2 \le 1$ for all k. Then, $\hat{\mathbf{g}}_k$ is a stochastic gradient for $\nabla f(\mathbf{x})$ with bias ϵ, and with variance $\sigma^2 + \delta + r(s)$.*

Proof. We first prove the bias bound. Fix any choice of k. Let us say the above procedure *succeeds* if the estimator c it produces satisfies $|c-(1-b_k\mathbf{a}_k^\top\mathbf{x})| \le \delta$, and say it fails otherwise.
There are two cases, depending on c. Suppose we have $c \in [-2\delta, 2\delta]$. Then, we have

$$\mathbb{E}[\hat{\mathbf{g}}_k \mid c \in [-2\delta,2\delta]] = \mathbb{E}[\mathbf{g}_k] = \mathbf{g}'_k\ ,$$

so obviously in this case we are unbiased.
On the other hand, if $c \notin [-2\delta, 2\delta]$, then we have

$$\mathbb{E}[\hat{\mathbf{g}}_k \mid c \notin [-2\delta,2\delta]] = (\mathbf{g}'_k + \mathbf{w}_k)\Pr[c \notin [-2\delta,2\delta], \text{success}] + \mathbf{g}_k \Pr[c \notin [-2\delta,2\delta], \text{failure}]\ , \tag{14}$$

where $\|\mathbf{w}_k\|_2 \leq O(\delta)$, since if $c \notin [-2\delta, 2\delta]$ and we succeed, this implies that $1 - b_k \mathbf{a}_k^\top \mathbf{x}_k \notin [\delta, \delta]$, and thus in this case

$$\begin{aligned}\|\mathbb{E}[\hat{\mathbf{g}}_k \mid c \notin [-2\delta, 2\delta], \text{success}] - \mathbf{g}'_k\| &= \left\| \left(\mathbb{E}[Q(1 - b_k \mathbf{a}_k^\top \mathbf{x})] - H(1 - b_k \mathbf{a}_k^\top \mathbf{x})\right)(-b_k \mathbf{a}_k)\right\| \\ &\leq \left|\mathbb{E}[Q(1 - b_k \mathbf{a}_k^\top \mathbf{x})] - H(1 - b_k \mathbf{a}_k^\top \mathbf{x}))\right| \\ &= \left|P(1 - b_k \mathbf{a}_k^\top \mathbf{x}) - H(1 - b_k \mathbf{a}_k^\top \mathbf{x})\right| \\ &\leq O(\delta) \ ,\end{aligned}$$

by assumption.

Finally, since $\Pr[c \notin [-2\delta, 2\delta], \text{failure}] \leq O(\delta)$, and $\|\mathbf{g}_k\|_2 = \|\mathbf{a}_k\|_2 \leq 1$ by assumption, (14) implies that $\|\mathbb{E}[\hat{\mathbf{g}}_k] - \mathbf{g}'_k\|_2 \leq O(\delta)$. By an appropriate choice of constants, this implies the desired bias bound for any fixed $\mathbf{a}_k$. Taking an expectation over all $\mathbf{a}_k$ yields the desired result.

We now turn our attention to the variance bound. We again split into two cases, depending on c. Clearly, we have

$$\mathbb{E}[\|\hat{\mathbf{g}}_k - \nabla f(\mathbf{x})\|_2^2 \mid c \in [-2\delta, 2\delta]] = \sigma^2 \ .$$

The interesting case is when $c \notin [-2\delta, 2\delta]$. In this case, we have

$$\begin{aligned}\mathbb{E}[\|\hat{\mathbf{g}}_k - \nabla f(\mathbf{x})\|_2^2 \mid c \notin [-2\delta, 2\delta]] &= \mathbb{E}[\|\hat{\mathbf{g}}_k - \nabla f(\mathbf{x})\|_2^2 \mid c \notin [-2\delta, 2\delta], \text{success}] \Pr[\text{success}] \\ &\quad + \mathbb{E}[\|\hat{\mathbf{g}}_k - \nabla f(\mathbf{x})\|_2^2 \mid c \notin [-2\delta, 2\delta], \text{failure}] \Pr[\text{failure}] \ ,\end{aligned}$$

as before. We analyze each term separately. As before, observe that if $c \notin [-2\delta, 2\delta]$ and we succeed, then $1 - b_k \mathbf{a}_k^\top \mathbf{x} \notin [-\delta, \delta]$. Hence, we have

$$\begin{aligned}\mathbb{E}[\|\hat{\mathbf{g}}_k - \nabla f(\mathbf{x})\|_2^2 \mid c \in [-2\delta, 2\delta], \text{success}] &= \mathbb{E}_{\mathbf{g}_k} \mathbb{E}_{\mathbf{a}_k}[\|\mathbf{g}_k - \nabla f(\mathbf{x})\|_2^2 \mid c \in [-2\delta, 2\delta], \text{success}] \\ &= \sigma^2 + \mathbb{E}_{\mathbf{a}_k}\left[\mathbb{E}_{\mathbf{g}_k}[\|\mathbf{g}_k - \mathbf{g}'_k\|_2^2 \mid c \in [-2\delta, 2\delta], \text{success}, \mathbf{a}_k]\right] \\ &\sigma^2 + \mathbb{E}_{\mathbf{a}_k}\left[\|\mathbf{w}_k\|_2^2 + \mathbb{E}_{\mathbf{g}_k}[\|\mathbf{g}_k\|_2^2 \mid c \in [-2\delta, 2\delta], \text{success}, \mathbf{a}_k]\right] \\ &\leq \sigma^2 + \delta^2 + r(s) \ .\end{aligned}$$

Moreover, since $(a + b)^2 \leq 2a^2 + 2b^2$, we have

$$\begin{aligned}\mathbb{E}[\|\hat{\mathbf{g}}_k - \nabla f(\mathbf{x})\|_2^2 \mid c \in [-2\delta, 2\delta], \text{failure}] &\leq \mathbb{E}[\|\hat{\mathbf{g}}_k\|_2^2 \mid c \in [-2\delta, 2\delta], \text{failure}] + 2 \\ &\leq r(s) + 2 \ .\end{aligned}$$

Hence the variance if $c \notin [-2\delta, 2\delta]$ is upper bounded by

$$\mathbb{E}[\|\hat{\mathbf{g}}_k - \nabla f(\mathbf{x})\|_2^2 \mid c \notin [-2\delta, 2\delta]] \leq \sigma^2 + \delta^2 + r(s) + \delta(1 + r(s)) \ ,$$

which simplifies to the claimed bound. □

20.3.2 A bound on the refetching probability

We now show that under a reasonable generative model, we will not have to refetch very often. Under this assumption, we show:

Theorem 7. *Suppose $\mathbf{a}_k$ are C-isotropic. Then, the probability we refetch at any iteration under the ℓ_2-refetching scheme is at most $\frac{R}{nC(1-\delta)^2} + O(\delta)$.*

Proof. Fix any $\mathbf{x}$ with $\|\mathbf{x}\|_2 \leq R$. By Lemma 11, the number of points with $1 - b_k \mathbf{a}_k^\top \mathbf{x} \in [-3\delta, 3\delta]$ is at most $\frac{R}{C(1-\delta)^2}$. If we succeed, and $1 - b_k \mathbf{a}_k^\top \mathbf{x} \notin [-3\delta, 3\delta]$, then by definition we do not refetch, the probability we refetch is bounded by the sum of the probability we choose a point with $1 - b_k \mathbf{a}_k^\top \mathbf{x} \in [-3\delta, \delta]$ and the probability of failure. By the above, this is bounded by $\frac{R}{nC(1-\delta)^2} + O(\delta)$, as claimed. □

20.4 ℓ_1-refetching

A simpler algorithmic to ensure we do not have any gradient flips is as follows. After getting the quantized sample $Q(\mathbf{a}_k)$, we can compute upper and lower bounds on $1 - b_k\mathbf{a}_k^\top\mathbf{x}$. The upper bound is given by:

$$1 - b_k Q(\mathbf{a}_k)^\top \mathbf{x} + \frac{\|\mathbf{x}\|_1}{s} ,$$

and the lower bound is given by:

$$1 - b_k Q(\mathbf{a}_k)^\top \mathbf{x} - \frac{\|\mathbf{x}\|_1}{s} ,$$

where $1/s$ is "resolution" of the quantization. If the upper and lower bounds of a quantized sample have the same sign, then we can be certain that no "flipping" will occur, and we can use the quantized sample. otherwise we send a request to fetch the original data and use it to compute the gradient. This seems to work well in practice, however, we could not prove any guarantees about how often we refetch with this guarantee, under any reasonable generative models.

21 Optimal Quantization Strategy

We prove Lemma 3 and Theorem 2 in the main paper here.

Problem Setting Assume a set of real numbers $\Omega = \{x_1, \ldots, x_N\}$ with cardinality N. WLOG, assume that all numbers are in $[0, 1]$ and sorted are sorted such that $x_1 \leq \ldots \leq x_N$.

The goal is to find an partition $\mathcal{I} = \{I_j\}_{j=1}^s$ of $[0, 1]$ into s disjoint intervals, so that if we randomly quantize every $x \in I_j$ to an endpoint of I_j, the variance is minimal over all possible partitions of $[0, 1]$ into k intervals. Formally:

$$\begin{aligned} \min_{\mathcal{I}:|\mathcal{I}|=s} \quad & \mathcal{MV}(\mathcal{I}) := \frac{1}{N}\sum_{j=1}^{k}\sum_{x_i \in I_j} \operatorname{err}(x_i, I_j) \\ \text{s.t.} \quad & \bigcup_{j=1}^{s} I_j = [0,1], \quad I_j \cap l_k = \emptyset \text{ for } k \neq j \end{aligned} \tag{15}$$

where $\operatorname{err}(x, I) = (b-x)(x-a)$ is the variance for point $x \in I$ if we quantize x to an endpoint of $I = [a, b]$. That is, $\operatorname{err}(x, I)$ is the variance of the (unique) distribution D supported on a, b so that $\mathbb{E}_{X\sim D}[X] = x$.

Given an interval $I \subseteq [0, 1]$, we let Ω_I be the set of $x_j \in \Omega$ contained in I. We also define $\operatorname{err}(\Omega, I) = \sum_{x_j \in I} \operatorname{err}(x_j, I)$. Given a partition $\mathcal{I}$ of $[0, 1]$, we let $\operatorname{err}(\Omega, \mathcal{I}) = \sum_{I \in \mathcal{I}} \operatorname{err}(\Omega, I)$. We let the optimum solution be $\mathcal{I}^* = \operatorname{argmin}_{|\mathcal{I}|=k} \operatorname{err}(\Omega, \mathcal{I})$, breaking ties randomly.

Lemma. *There is an $\mathcal{I}^*$ so that all endpoints of any $I \in \mathcal{I}^*$ are in $\Omega \cup \{0, 1\}$.*

Proof. Fix any endpoint b of intervals in $\mathcal{I}^*$. WLOG assume that $b \neq 0, 1$. Then we must have $I = [a, b]$ and $I' = [b, c]$ for some $I, I' \in \mathcal{I}^*$. Observe that the choice of b only affects the error for points in $I \cup I'$. We have that $\operatorname{err}(\Omega, I) + \operatorname{err}(\Omega, I')$ is given by

$$\begin{aligned} &\sum_{x \in I}(b-x)(x-a) + \sum_{x \in I'}(c-x)(x-b) \\ &= Ab + C , \end{aligned}$$

where A, C are constants which do not depend on b. Hence, this is a linear objective in b. Since b can freely range between the rightmost point in I and the leftmost point in I', there is an optimizer for this solution at one of those two points. Hence we may choose $b \in \Omega$. □

Therefore, to solve the problem in an exact way, we just need to select a subset of data points in Ω as quantization points. Define $T(k, m)$ be the optimal total variance for points in $[0, d_m]$ with k quantization levels choosing $d_m = x_m$ for all $m = 1, 2, \cdots, N$. Our goal is to calculate $T(s, N)$. This problem can be solved by dynamic programing using the following recursion

$$T(k, m) = \min_{j \in \{k-1, k, \cdots, m-1\}} T(k-1, j) + V(j, m),$$

where $V(j, m)$ denotes the total variance of points falling into $[d_j, d_m]$. The complexity of calculating the matrix $V(\cdot, \cdot)$ is $O(N^2 + N)$ and the complexity of calculating matrix $T(\cdot, \cdot)$ is $O(kN^2)$. The memory cost is $O(kN + N^2)$.

21.1 Heuristics

The exact algorithm has a complexity that is quadratic to the number of data points. To make our algorithm practical, we develop an approximation algorithm that only needs to scan all data points once and has linear complexity to N.

Discretization We can discretize the range $[0, 1]$ into M intervals, i.e., $[0, d_1), [d_1, d_2), \cdots, [d_{M-1}, 1]$ with $0 < d_1 < d_2 < \cdots < d_{M-1} < 1$. We then restrict our algorithms to only choose k quantization points within these M points, instead of all N points in the exact algorithm. The following result bounded the quality of this approximation.

Theorem. *Let the maximal number of data points in each "small interval" (defined by $\{d_m\}_{m=1}^{M-1}$) and the maximal length of small intervals be bounded by bN/M and a/M, respectively. Let $\mathcal{I}^* := \{l_j^*\}_{k=1}^{k-1}$ and $\hat{\mathcal{I}}^* := \{\hat{l}_k^*\}_{k=1}^{k-1}$ be the optimal quantization to* (15) *and the solution with discretization. Let cM/k be the upper bound of the number of small intervals crossed by any "large interval" (defined by $\mathcal{I}^*$). Then we have the discretization error bounded by*

$$\mathcal{MV}(\hat{\mathcal{I}}^*) - \mathcal{MV}(\mathcal{I}^*) \leq \frac{a^2 bk}{4M^3} + \frac{a^2 bc^2}{Mk}.$$

Proof. Let p_0^* be 0 and $p_K^* = 1$.We quantitize $\{p_k^*\}_{k=1}^{K-1}$ one element by one element, while monitor the changing of the total variance $N \times \mathcal{MV}(\cdot)$. We first quantize p_1^* to the closest value (denoted it by $Q(p_1^*)$) in $\{d_m\}_{m=1}^{M-1} \cup \{p_0^*, p_K^*\}$ under the monotonicity constraint, that is, $p_0^* \leq Q(p_1^*) \leq p_2^*$. Here, one important observation is $|p_1^* - Q(p_1^*)| \leq a/M$. Consider the total variance of this new solution $Q(p_1^*), p_2^*, \cdots, p_{K-1}^*$. The variance of points falling into the range $[p_2^*, 1]$ does not change at all. Without the loss of generality, assume $p_1^* \geq Q(p_1^*)$.

Next we consider points falling into the following three sets $C_1 = [p_0^*, Q(p_1^*)]$, $C_2 = [Q(p_1^*), p_1^*]$, and $C_3 = [p_1^*, p_2^*]$. The variance of points of falling into C_1 gets reduced from the form of variance in (21). Next we only need to check the variance change for points in C_2 and C_3. Consider C_2 first. The variance for point x in C_2 is

$$(x - Q(p_1^*))(p_1^* - x) \leq \frac{a^2}{4M^2}.$$

Thus, the change of variance for points in C_2 would be bounded by $\frac{a^2}{4M^2}$. Then consider C_3. The change of variance for point x in C_3 is

$$\begin{aligned}
&(x - Q(p_1^*))(p_2^* - x) - (x - p_1^*)(p_2^* - x) \\
=&(p_1^* - Q(p_1^*))(p_2^* - x) \\
\leq&\frac{a}{M}(p_2^* - x)
\end{aligned}$$

Therefore, the change of total variance from $\{p_1^*, p_2^*, \cdots, p_{K-1}^*\}$ to $\{Q(p_1^*), p_2^*, \cdots, p_{K-1}^*\}$ is bounded by

$$\begin{aligned}
&\sum_{x \in C_2} \frac{a^2}{4M^2} + \sum_{x \in C_3} \frac{a}{M}(p_2^* - x) \\
\leq&\frac{Nb}{M}\frac{a^2}{4M^2} + \frac{a}{M}\frac{Nb}{M}\sum_{t=1}^{cM/K} t\frac{a}{M} \\
\leq&\frac{a^2 bN}{4M^3} + \frac{a^2 bc^2 N}{MK^2} \qquad (16)
\end{aligned}$$

Similarly, we quantitize p_2^2 in $\{Q(p_1^*), p_2^*, \cdots, p_{K-1}^*\}$ to get a new solution $\{Q(p_1^*), Q(p_2^*), \cdots, p_{K-1}^*\}$ while maintain the monotonicity. We can establish the same upper bound to (16). Following this idea, we can obtain a quantization solution $\{Q(p_1^*), Q(p_2^*), \cdots, Q(p_{K-1}^*)\}$. Therefore, we obtain that

$$\begin{aligned}&\mathcal{MV}(Q(p_1^*), Q(p_2^*), \cdots, Q(p_{K-1}^*)) - \mathcal{MV}(p_1^*, \cdots, p_{K-1}^*)\\ &\leq \frac{a^2bK}{4M^3} + \frac{a^2bc^2}{MK}.\end{aligned}$$

Using the fact that $\mathcal{MV}(p_1^*, \cdots, p_{K-1}^*)$ is smaller than $\mathcal{MV}(Q(p_1^*), Q(p_2^*), \cdots, Q(p_{K-1}^*))$ proves the claim. $\square$

Theorem 2 suggests that the mean variance using the discrete variance-optimal quantization will converge to the optimal with the rate $O(1/Mk)$.

22 A Greedy Algorithm for Finding Quantization Points

22.1 Setup

We have n points $\Omega = x_1, \ldots, x_n \in [0, 1]$. Our goal is to partition $[0, 1]$ into k intervals $I_1, \ldots, I_k$, so that if we quantize all x_i in I_j to an endpoint of I_j, we minimize the variance. If $I = [a, b]$, and $x \in I$, it is not hard to show that the variance of the quantization is given by $\mathrm{err}(x, I) = (b - x)(x - a)$.

Notation Given an interval $I \subseteq [0, 1]$, we let Ω_I be the set of $x_j \in \Omega$ contained in I. We also define $\mathrm{err}(\Omega, I) = \sum_{x_j \in I} \mathrm{err}(x_j, I)$. Given a partition $\mathcal{I}$ of $[0, 1]$, we let $\mathrm{err}(\Omega, \mathcal{I}) = \sum_{I \in \mathcal{I}} \mathrm{err}(\Omega, I)$. We also let $\mathcal{I}^* = \mathrm{argmin}_{|\mathcal{I}|=k} \mathrm{err}(\Omega, \mathcal{I})$ (if there are multiple, then choose one arbitrarily), and we let $\mathrm{OPT}_k = \mathrm{err}(\Omega, \mathcal{I}^*)$.

We require the following lemma, whose proof is trivial and omitted.

Lemma 12. *If $I \subseteq I'$, then $\mathrm{err}(\Omega, I) = \mathrm{err}(\Omega_I, I) \leq \mathrm{err}(\Omega_I, I')$.*

22.2 A nearly linear time algorithm for nearly minimizing the error

First, we observe that it is trivial that the optimal partition must have endpoints solely at points in Ω. Thus we may restrict our attention to such partitions. The algorithm is given in Algorithm 1. The algorithm is inspired by greedy merging algorithms for histogram recovery.

We first show this algorithm runs in nearly linear time:

Theorem 8. *Given any $\Omega, k, \gamma, \delta$, we have that* $\textsc{AdaQuant}(\Omega, k, \gamma, \delta)$ *runs in time $O(n(\log(n/\gamma) + \log\log 1/\delta))$*

Our main contribution is to show that the algorithm does indeed achieve good error:

Theorem 9. *Given any $\Omega, k, \gamma, \delta$, let $\mathcal{I}$ be the output of* $\textsc{AdaQuant}(\Omega, k, \gamma, \delta)$. *Then we have that $\mathrm{err}(\Omega, \mathcal{I}) \leq \left(1 + \frac{1}{\gamma}\right) \mathrm{OPT}_k$.*

Proof. Partition $\mathcal{I} = \mathcal{F} \cup \mathcal{J}$, where $\mathcal{F}$ is the set of intervals $I \in \mathcal{I}$ so that $I \subseteq I'$ for some $I' \in \mathcal{I}^*$, and let $\mathcal{J}$ be the remaining intervals. Observe that by a simple counting argument, we have that $|\mathcal{J}| \leq k$. By Lemma 12, we have that

$$\sum_{I \in \mathcal{F}} \mathrm{err}(\Omega, I) \leq \mathrm{OPT}_k \ .$$

We now seek to bound the error along intervals in $\mathcal{J}$. Let $I \in \mathcal{J}$. It must have been merged in some iteration of $\textsc{AdaQuant}$. Therefore, in that iteration, there must have been $(1 + \gamma)k$ merged intervals $J_1, \ldots, J_{(1+\gamma)k}$ so that $\mathrm{err}(\Omega, I) \leq \mathrm{err}(\Omega, J_\ell)$, for all $\ell = 1, \ldots, (1 + \gamma)k$. By a simple counting argument, at most k of the J_ℓ are not contained in some interval in $\mathcal{I}^*$. WLOG, assume that $J_1, \ldots, J_{\gamma\ell}$ are all contained in some interval of $\mathcal{I}^*$. By

Lemma 12, we have that $\sum_{j=1}^{\gamma\ell} \mathrm{err}(\Omega, J_j) \leq \mathrm{OPT}$. In particular, this implies that $\mathrm{err}(\Omega, I) \leq \mathrm{OPT}/(\gamma k)$. Since this holds for all $I \in \mathcal{J}$, and $|\mathcal{J}| \leq k$, this implies that

$$\sum_{I \in \mathcal{J}} \mathrm{err}(\Omega, I) \leq k \frac{\mathrm{OPT}_k}{\gamma k} \leq \frac{1}{\gamma} \mathrm{OPT}_k \ .$$

Combining the above two expressions yields that $\mathrm{err}(\Omega, \mathcal{I}) \leq \left(1 + \frac{1}{\gamma}\right) \mathrm{OPT}_k$, as claimed. □

23 Implementation on FPGA

In this section, we show the computation pipeline for full-precision SGD and quantized SGD implemented on FPGA in figure 18 and 19. The FPGA implementation work is submitted to IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM) 2017.

First, we introduce the target platform, Intel Xeon+FPGA. Then, we present our FPGA implementations for SGD that work on 32-bit floating point data and lower precision quantized data, respectively. Under each subsection, we discuss the trade-off for the circuit design we faced during the implementation phase.

23.1 Target platform: Intel Xeon+FPGA

Our target platform is the Intel Xeon+FPGA represented in Figure 17, made available through the *Intel-Altera Heterogeneous Architecture Research Platform*[4]. It combines a 10-core CPU (Intel Xeon E5-2680 v2, 2.8 GHz) and an FPGA (Altera Stratix V 5SGXEA) on a 2-socket motherboard. The FPGA has cache-coherent access to the CPU-connected main memory through QPI (Quick Path Interconnect) with a combined read and write bandwidth of around 6.5 GB/s. The usage of an FPGA accelerator, that we implement using VHDL, is pretty straightforward with Intel-provided software libraries, a QPI endpoint and an FPGA based page table of our own implementation: At the start of an application the required amount of shared memory is allocated by software in 4MB pages and the page table on the FPGA is populated with corresponding addresses. The accelerator can request cache-line wide (64B) reads and writes to the entire memory through the use of the page table and the QPI endpoint, which in addition to handling the QPI protocol also implements an FPGA local cache (128KB two-way associative) using BRAMs. Apart from the page-table, a further addition we made to the interface is a reorder buffer, since by default QPI requests arrive out of order. In addition to an accelerator requesting data from QPI, the software can also issue direct writes, a useful feature for configuring some registers containing runtime parameters.

23.2 FPGA-SGD on float data

We first present an SGD implementation on the FPGA, that works on 32-bit floating-point data, which is very often the original data representation in machine learning data sets. In Figure 18 the computation pipeline for *float* FPGA-SGD is presented. As the data access width is a 64B cache-line on Xeon+FPGA, the circuit is designed to work on that data width. It is able to accept a cache-line at every clock cycle ($f_{clock} = 200MHz$), resulting in an internal processing rate of 12.8 GB/s.

Scale to # features: The challenging part of the design is to make it capable of handling a number of features D that is larger than 16, which is the default width of the pipeline. In fact, this is possible since all vector algebra can be performed iteratively, where each chunk contains 16 values. This implies that the number D needs to be a multiple of 16. To ensure that, we use zero-padding on a data set, if $D \bmod 16 \neq 0$. Thus, we can calculate how many cache-lines it takes for $\mathbf{a_i}$ (one row in the set) to be completely received:

$$\#\mathbf{a_i}\text{cache-lines} = \begin{cases} D/16 & \text{if } D \bmod 16 = 0 \\ \frac{D + (16 - D \bmod 16)}{16} & \text{if } D \bmod 16 \neq 0 \end{cases}$$

[4]Following the Intel legal guidelines on publishing performance numbers we want to make the reader aware that results in this publication were generated using preproduction hardware and software, and may not reflect the performance of production or future systems.

Algorithm 1 Nearly-linear time algorithm for finding quantization points

```
1: function AdaQuant($\Omega, k, \gamma, \delta$)
2:     Let $\mathcal{I} = [0,1]$ be a partition of $[n]$, initially with one breakpoint at each point in $\Omega \cup \{0,1\}$.
3:     while $|\mathcal{I}| > 2(1+\gamma)k + \delta$ do
4:         Pair up consecutive intervals in $\mathcal{I}$ to form $\mathcal{J}$
5:         for each $I \in \mathcal{J}$ do
6:             Let $e_I = \mathrm{err}(\Omega, I)$
7:         Let $\mathcal{J}_1$ be the set of $(1+\gamma)k$ intervals $I \in \mathcal{I}$ with largest $e_I$.
8:         for each $I \in \mathcal{J}_1$ do
9:             Let $I = I_1 \cup I_2$, where $I_1, I_2 \in \mathcal{I}$
10:            Remove $I$ from $\mathcal{J}$
11:            Insert $I_1, I_2$ into $\mathcal{J}$
12:        Let $\mathcal{I} \leftarrow \mathcal{J}$
13:    return the partition with endpoints at $\mathcal{I}$.
```

The only parameter that the design needs to scale for is the maximum number of dimensions D_{max} that can be supported, and the only value growing with that is the amount of BRAM for storing the model $\mathbf{x}$. We choose D_{max} to be 8192, since this is more than enough for most existing linear dense model training examples. The design can handle any number of samples, *N*, since training is done in a streaming fashion, as explained in the following.

Walk-through of computation pipeline: In the following, we explain what happens at each stage of the computation pipeline. The first stage is the dot product $\mathbf{a_i} \cdot \mathbf{x} = \sum_{j=1}^{D} a_{i,j} x_j$. When a cache-line containing a part of vector $\mathbf{a_i}$ arrives, it is first multiplied (①) with the corresponding part of vector $\mathbf{x}$. The multiplication is performed in floating point with multiplier IPs[5], which have 5 cycles latency and a result per cycle throughput. Then, the values are converted to a 32-bit integer (②) by multiplication with a large constant that is configurable during runtime, depending on the value range or the precision that is desired to be kept. This *float2fixed* conversion takes 1 cyle. After that, an adder tree (③) accumulates 16 values, each layer taking 1 cycle. At the output of the adder tree is a single accumulator (④). It accumulates the results coming out from the adder tree for the pre-calculated number of #$\mathbf{a_i}$cache-lines, thus building the final value for the dot product. The dot product result is converted back to floating point (⑤), since the next stages of the calculation will be performed with floating point data. In the next part, the rest of the gradient calculation takes place. First, the scalar value b, which is the true inference value in the data set, is subtracted from the dot product (⑥) using a floating point adder IP, which has 7 cycles latency and a result per cycle throughput. As for where the b value comes from, it is actually received some cycles before the subtraction takes place and is put into a FIFO (Ⓑ), waiting there until the dot product result is ready. After the subtraction, a floating point multiplication takes place with the step size γ (⑦), which can be configured to any *float* value during runtime. At the end of this step, we have a scalar value $\gamma(\mathbf{a_i} \cdot \mathbf{x} - b_i)$, which needs to be multiplied with vector $\mathbf{a_i}$. At this stage, a FIFO (Ⓐ) already contains all parts of vector $\mathbf{a_i}$, because the incoming cache-lines are written to this FIFO simultaneously as they were sent to the dot product calculation. The scalar-vector multiplication (⑧) takes place in floating point, where all parts of $\mathbf{a_i}$ are multiplied with the same scalar value. This gives us the gradient $\mathbf{g_i}$ one part at a time, which undergoes *float2fixed* conversion (⑨), so that a cycle-by-cycle update of the model $\mathbf{x}$ (Ⓒ) can take place, which would not be possible with a floating point adder having 7 cycles latency. The part of the gradient, which is ready, is applied to the corresponding part of the model. After the last part of the gradient is subtracted from the model, the update for $\mathbf{a_i}$ (a row in the data set) is completed. After all rows go through the same calculation, one so called epoch is completed. In Figure 18 we can see that the model we apply the updates to and the model we read for the dot product are separate. Only when a certain batch size (the number of already processed $\mathbf{a_i}$) is reached, the updated model is carried on to the actual model (Ⓓ). The reason for this is the latency introduced by the computation pipeline: In theory, the whole gradient calculation and the update to the model should be an atomic operation. However, since we would like to exploit the deep-pipelining an FPGA provides to achieve high processing throughput, we don't perform this operation atomically. Instead, we keep the actual model and the updated model separate and only carry out the accumulated update when

[5]All floating point IPs are created via Altera Quartus II 13.1

Table 2: Choice of quantization levels, lower and upper bounds, so that only integer values are produced.

Levels	Data set positive	Data set negative	Needed bits
s=2	$[L, U] = [0, 1]$	N/A	1, *Q1*
s=3	$[L, U] = [0, 2]$	$[L, U] = [-1, 1]$	2, *Q2*
s=9	$[L, U] = [0, 8]$	$[L, U] = [-4, 4]$	4, *Q4*
s=129	$[L, U] = [0, 128]$	$[L, U] = [-64, 64]$	8, *Q8*

a certain batch size is reached (called also a mini-batch SGD). The batch size is a runtime configurable parameter, which should be set to the latency of the pipeline (36 cycles) to avoid any so called stale updates.
Trade-off: End-to-end float vs. hybrid computation We choose a hybrid (float+fixed) over end-to-end float computation because of the following reasons: (1) One floating point addition takes 7 cycles, resulting in a high latency adder tree (③). Because of the increased latency (as explained previously) a larger batch-size is required to avoid staleness, which slows down the convergence rate. (2) Again, since a floating point addition takes 7 cycles, a cycle-by-cycle accumulation (⑦) would not be possible, since the result of an ongoing addition is required in the next cycle. Thus, for keeping the processing rate at 64B/cycle, a hybrid design is chosen.

23.3 FPGA-SGD on quantized data

We explain how the FPGA-SGD design for *float* data is adjusted to work on quantized data. The main purpose of this design is simple: Instead of reading *float* data, where only 16 values can be fit into a cache-line, we would like to quantize the data beforehand, so that we can fit more than 16 values into a cache-line, thus utilize the available bandwidth more efficiently and gain speedup. There is one main challenge in making the FPGA-SGD work on quantized data: Scaling out the computation pipeline presented in Figure 18, so that it can work on more than 16 values in parallel. Before we dive in and explain how this is achieved, first we would like to give an overview of quantization options we consider and how the data layout looks like.
Quantization for FPGA-SGD: When we analyze what happens when we quantize data, given a non-integer value, the quantization still might produce a non-integer value. However, floating point arithmetic induced by non-integer values are hard to implement on the FPGA, and scaling out such a design would not be possible. We would like to take advantage of the fact that we can select the quantization variables $[L, U]$ and s intelligently, so that only integer values are produced. In Table 2 we present choices for these values.

After we have selected a quantization precision (one of *Q1*, *Q2*, *Q4* or *Q8*), the data set (the values in matrix $\mathbf{a}$) must be normalized to selected quantization's corresponding $[L, U]$. At this stage, the sign of the data set is considered for the normalization. For example, we choose not the normalize a negative data set into a positive interval, in order to keep existing zeros, thus we don't use *Q1* for negative data sets.
The layout of quantized data For calculating a correct gradient we need 2 quantization samples of the same data point. That is, if we for example select *Q8*, a quantized sample has 8 bits and we need 2 of them for calculating the gradient; the actual amount of bits we use is 16. That's why, when we perform quantization on a data set, we always create 2 samples and store them in memory next to each other. Thus, we can calculate how many quantized values can fit into one cache-line, a value we call K, in Table 3. The value K dictates the amount of zero-padding we need to perform, in order to be cache-line aligned, similar to as it did for *float* FPGA-SGD. Thus, the number of cache-lines required to receive one quantized row $Q(\mathbf{a_i})$ can be calculated:

$$\#\mathbf{a_i}\text{cache-lines} = \begin{cases} D/K & \text{if } D \bmod K = 0 \\ \frac{D+(K-D \bmod K)}{K} & \text{if } D \bmod K \neq 0 \end{cases} \tag{17}$$

Computation pipeline for quantized data: The computation pipeline for quantized data is presented in Figure 19. The selection of *Qx*, which determines the width of the pipeline is a generic parameter that can be set before synthesis. So, for each *Qx* a different bitstream needs to be compiled. The pipeline is very similar to the *float* version, explained in Section 23.2. For this reason, our explanation here only focuses on the differences and how the pipelines are scaled out. The first thing to note is that *Qx* pipelines are working only on integer data, so there is no need for converters.

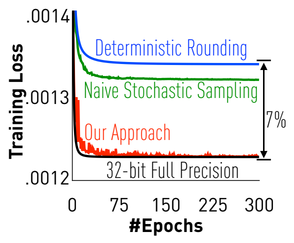
Table 3: Number of received values in a single cache-line.

Data type	*Q1*	*Q2*	*Q4*	*Q8*	*float*
Number of values in a cache-line, K	256	128	64	32	16

This is because, the arriving quantized data $Q(\mathbf{a_i})$ is already in integer form, as explained previously, and the inference values b are also converted to integer by multiplication with a large constant. Another difference here is that for a given value in vector $\mathbf{a_i}$, 2 quantized samples arrive because of the double sampling method. The first sample is given to the dot product calculation (①) and the second sample is put into a FIFO (Ⓐ), where it is kept until the dot product result is ready, as depicted in Figure 19. The last difference we would like to mention is the appliance of the step size γ (②), which is actually a division. Since we are dealing with integer data here, we choose to apply γ as a bit-shift operation. By how many bits the value is shifted to right is a runtime configurable parameter, allowing adjustments according to data set characteristics.

Scaling out for quantized data and trade-offs: Now, scaling out the pipeline for *Q8* and *Q4* is very straightforward as conventional signed multipliers (③), implemented by DSP resources, followed by a bitshift, to keep the data width at 32 bits, is used as shown in Figure **??**. However, for *Q2* and *Q1*, we can do a much more efficient multiplication using multiplexers as shown in **??**, since one of the multiplicands is only 2 bits and 1 bit, respectively. Doing this efficient multiplication allows *Q2* pipeline to scale to 128 value parallelism, which would have otherwise required a 100% usage of the available DSP resources on the target FPGA (see Table **??**). However, the pipeline shown in Figure 19a does not scale to 256 value parallelism, even though *Q1* multiplier is just one multiplexer. The main problem here is that the adder tree is too wide and deep, making the routing very challenging. Once it has become clear that we cannot scale the pipeline to this parallelism, we decided to halve it to process *Q1* data. For doing that, we split an arriving cache-line into 2 parts (④), which are processed sequentially by the pipeline shown in Figure 19b. The processing rate of this pipeline is 32B/cycle, or 6.4 GB/s, which is slightly less than the memory bandwidth. Thus, *Q1* FPGA-SGD will actually be compute bound.

¿¿¿¿¿¿¿ master



(**b**)

=======

3.4 Double Sampling for Unbiased Stochastic Gradient

The naive way to use low-precision samples $\hat{\mathbf{a}}_t := Q(\mathbf{a}_t)$ is

$$\hat{\mathbf{g}}_t := \hat{\mathbf{a}}_t \hat{\mathbf{a}}_t^\top \mathbf{x} - \hat{\mathbf{a}}_t b_t.$$

However, *the naive approach does not work* (that is, it does not guarantee convergence), because it is biased:

$$\mathbb{E}[\hat{\mathbf{g}}_t] := \mathbf{a}_t \mathbf{a}_t^\top \mathbf{x} - \mathbf{a}_t b_t + D_\mathbf{a} \mathbf{x},$$

where $D_\mathbf{a}$ is diagonal and its ith diagonal element is

$$\mathbb{E}[Q(\mathbf{a}_i)^2] - \mathbf{a}_i^2.$$

¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ master Since $D_\mathbf{a}$ is non-zero, we obtain a *biased* estimator of the gradient, so the iteration is unlikely to converge. The figure on the right illustrates the bias caused by a non-zero $D_\mathbf{a}$. In fact, it is easy to see that in instances where the minimizer $\mathbf{x}$ is large and gradients become small, we will simply diverge.

We now present a simple method to fix the biased gradient estimator. We generate two independent random quantizations and revise the gradient:

$$\mathbf{g}_t := Q_1(\mathbf{a}_t)(Q_2(\mathbf{a}_t)^\top \mathbf{x} - b_t)\ . \tag{6}$$

This gives us an unbiased

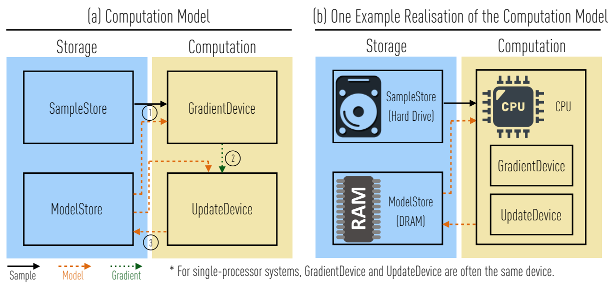


Figure 2: A schematic representation of the computational model.
=======

(–) Non-Linear Models. We extend our results to non-linear models, such as SVM. We can stretch our multiple-sampling strategy to provide unbiased estimators for any polynomials, at the cost of increased variance. Building further, we employ Chebyshev polynomials to approximate the gradient of *arbitrary smooth loss functions* within arbitrarily low bias, and to provide bounds on the error of an SGD solution obtained from low-precision samples.
Further, we examine whether this approach can be applied to non-smooth loss functions, such as SVM. We find that the machinery described above does not apply, for fundamental reasons. We use ideas from streaming and dimensionality reduction to develop a variant that is provably correct for non-smooth loss functions. We can show that, under reasonable assumptions, the added communication cost of supporting non-smooth functions is negligible. In practice, using this technique we are able to go as low as 8-bit precision for SVM and logistic regression. However, we notice that the straw man approach, which applies naive stochastic rounding over the input data to just 8-bit precision, converges to similar results, without the added complexity. This negative result is explained by the fact that, to approximate non-linearities such as the step function or the sigmoid well, our framework needs both high degree Chebyshev polynomials and relatively large samples.
¡¡¡¡¡¡¡ HEAD

2 Linear Models

In this section, we focus on linear models with possibly non-smooth regularization. We have labeled data points $(\mathbf{a}_1, b_1), (\mathbf{a}_2, b_2), \ldots, (\mathbf{a}_K, b_K) \in \mathbb{R}^n \times \mathbb{R}$, and our goal is to minimize the function =======

3 Linear Models

In this section, we focus on linear models with possibly non-smooth regularization. We have labeled data points $(\mathbf{a}_1, b_1), (\mathbf{a}_2, b_2), \ldots, (\mathbf{a}_K, b_K) \in \mathbb{R}^n \times \mathbb{R}$, and our goal is to minimize the function ¿¿¿¿¿¿¿ master

$$F(\mathbf{x}) = \underbrace{\frac{1}{K}\sum_{k=1}^{K}\|\mathbf{a}_k^\top \mathbf{x} - b_k\|_2^2}_{=:f(\mathbf{x})} + R(\mathbf{x}) \ , \tag{3}$$

i.e., minimize the empirical least squares loss plus a non-smooth regularization $R(\cdot)$ (e.g., ℓ_1 norm, ℓ_2 norm, and constraint indicator function). SGD is a popular approach for solving large-scale machine learning problems. It works as follows: at step $\mathbf{x}_t$, given an unbiased gradient estimator $\mathbf{g}_t$, that is, $\mathbb{E}(\mathbf{g}_t) = \nabla f(\mathbf{x}_t)$, we update $\mathbf{x}_{t+1}$ by

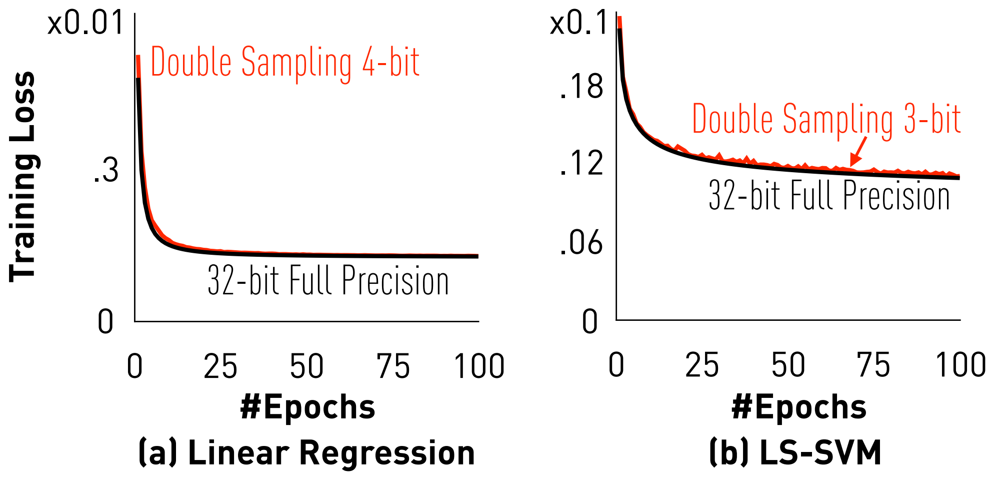
$$\mathbf{x}_{t+1} = \text{prox}_{\gamma_t R(\cdot)}\left(\mathbf{x}_t - \gamma_t \mathbf{g}_t\right),$$

where γ_t is the predefined step length. SGD guarantees the following convergence property:

Theorem 1. *[e.g., **?**, Theorem 6.3] Let the sequence $\{\mathbf{x}_t\}_{t=1}^T$ be bounded. Appropriately choosing the steplength, we have the following convergence rate for* (3)*:*

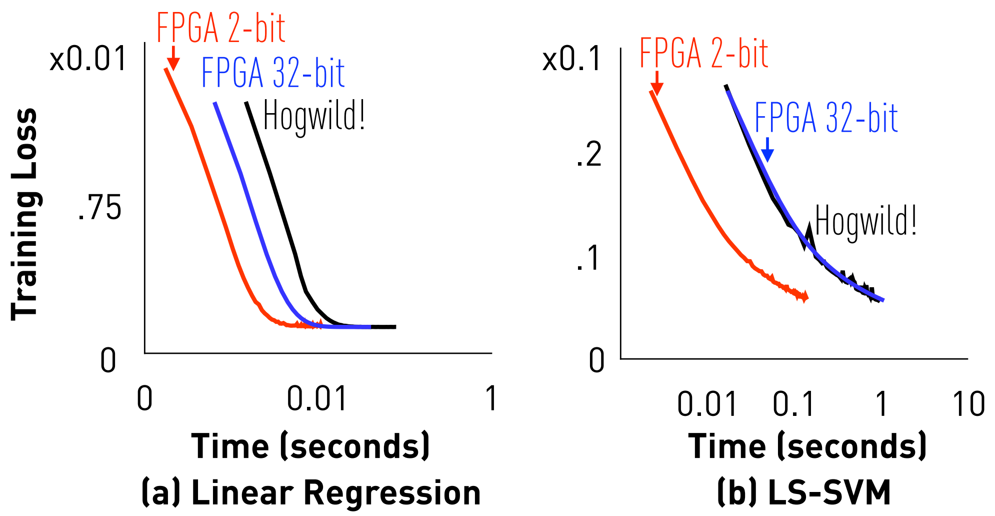
$$F\left(\frac{1}{T}\sum_{t=0}^{T}\mathbf{x}_t\right) - \min_{\mathbf{x}} F(\mathbf{x}) \leq \Theta\left(\frac{1}{T} + \frac{\sigma}{\sqrt{T}}\right) \tag{4}$$

where σ is the upper bound of the mean variance



¡¡¡¡¡¡¡ HEAD

Figure 7: Linear models with end-to-end low precision.



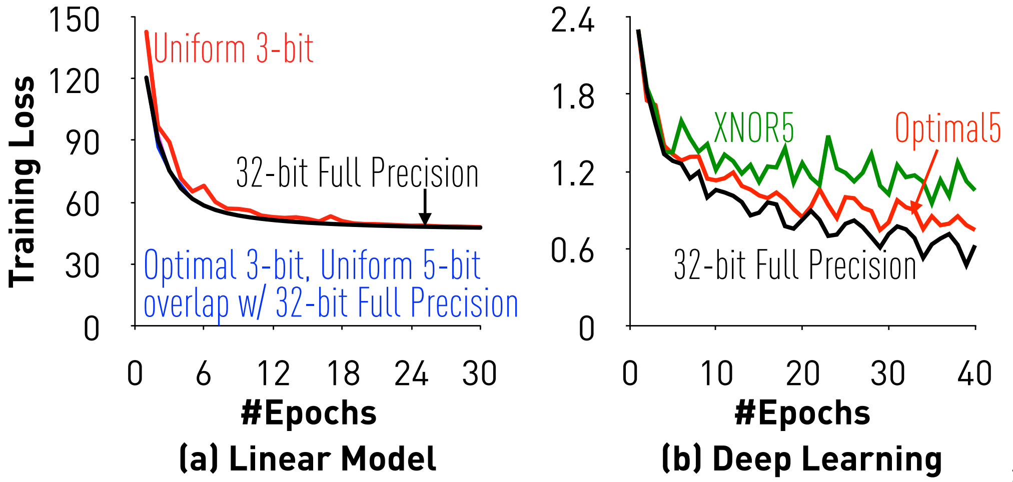
¡¡¡¡¡¡¡ HEAD

Figure 8: FPGA implementation of linear models.

=======

Figure 9: FPGA implementation of linear models.

¿¿¿¿¿¿¿ master



¡¡¡¡¡¡ HEAD

Figure 10: Optimal quantization strategy.

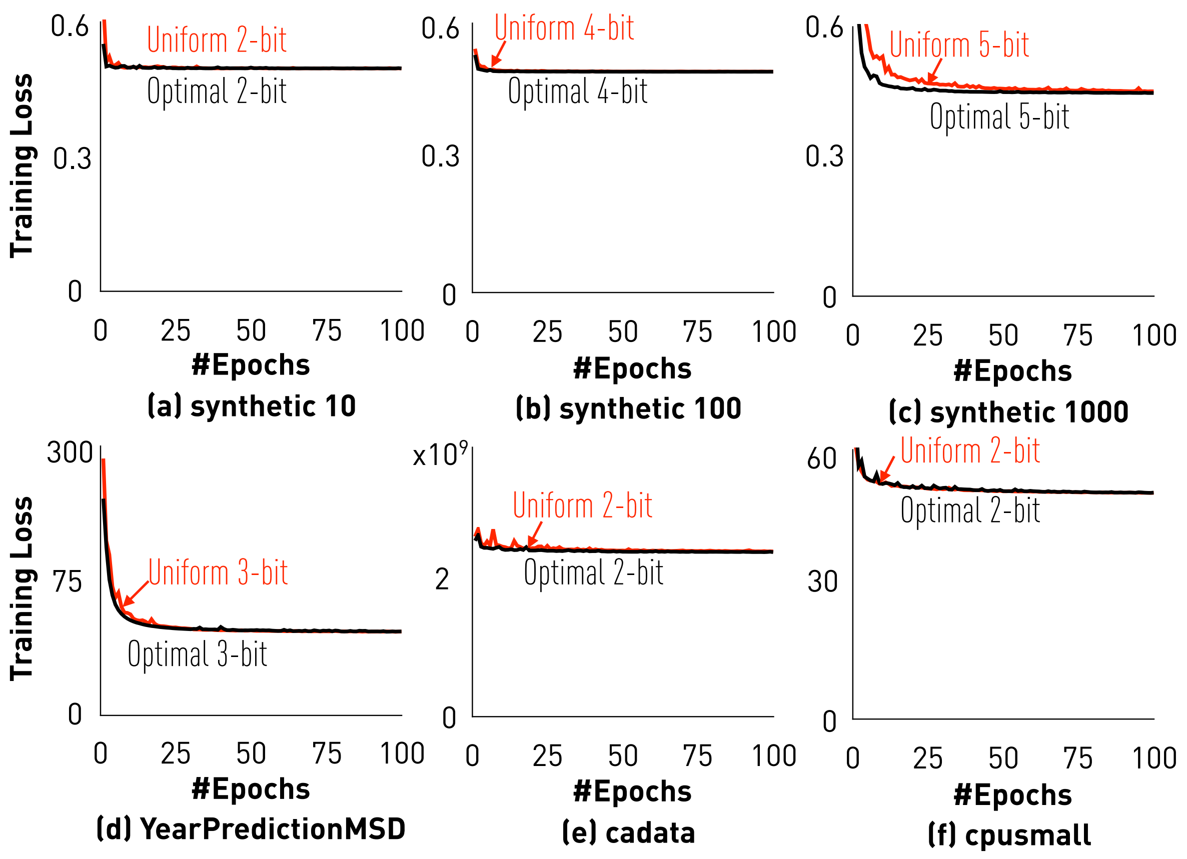


Figure 11: Linear regression with *quantized data* on multiple datasets

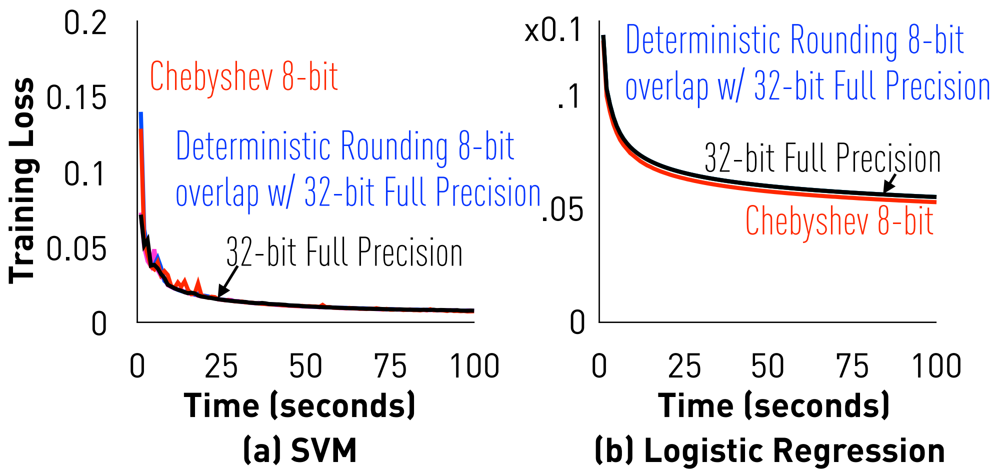


Figure 12: Non-linear models with Chebyshev approximation.

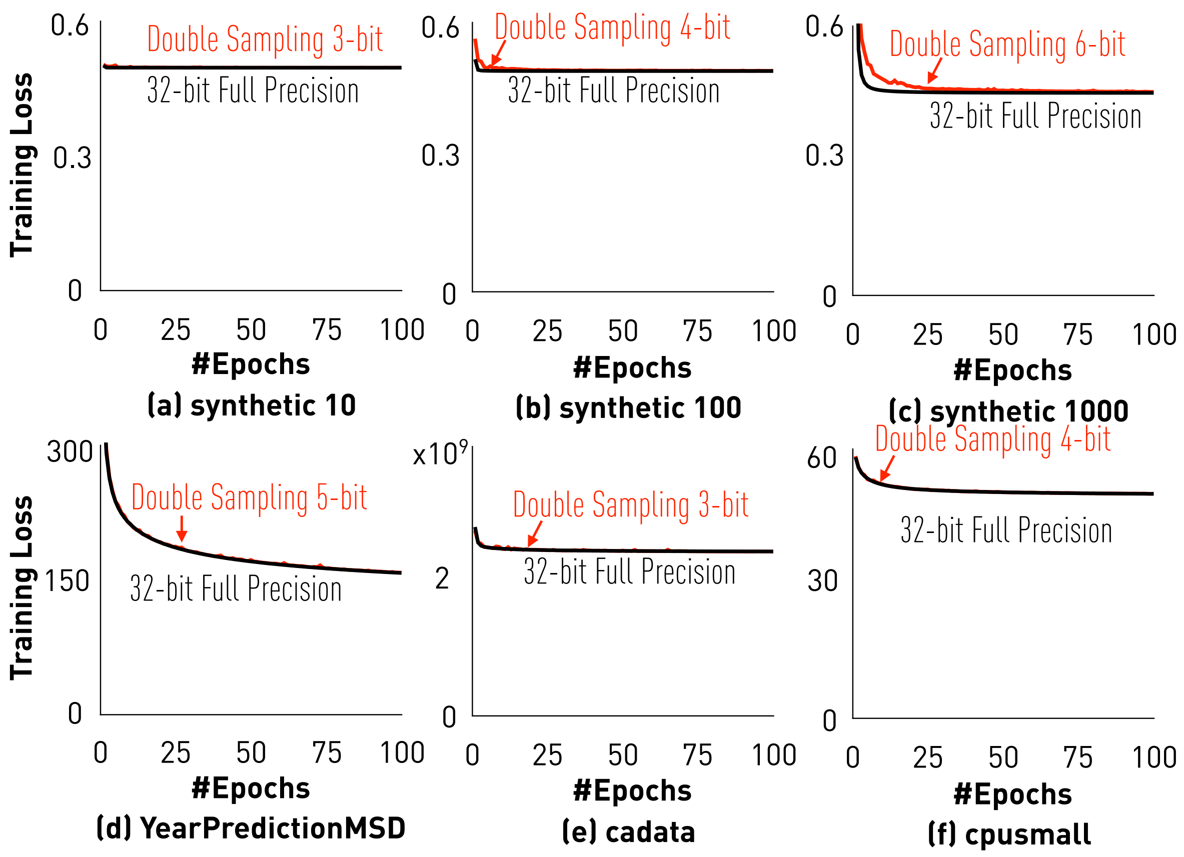


Figure 13: Linear regression with *end-to-end quantization* on multiple datasets

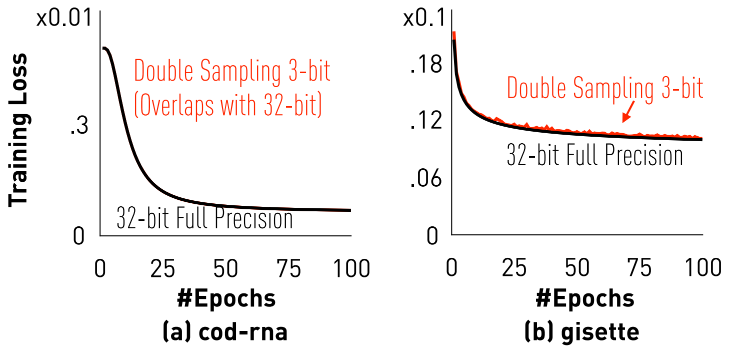


Figure 14: Least squares SVM with *end-to-end quantization* on multiple datasets

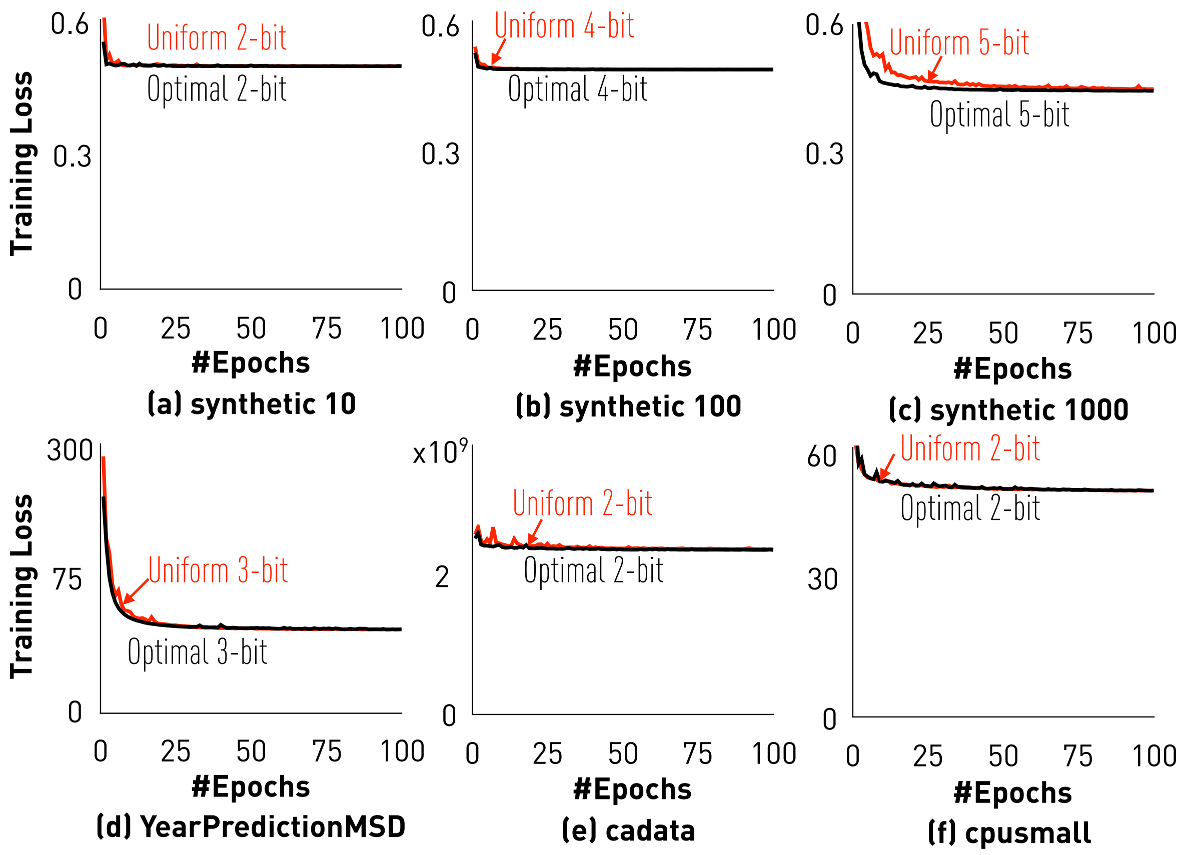


Figure 15: Linear regression with *quantized data* on multiple datasets

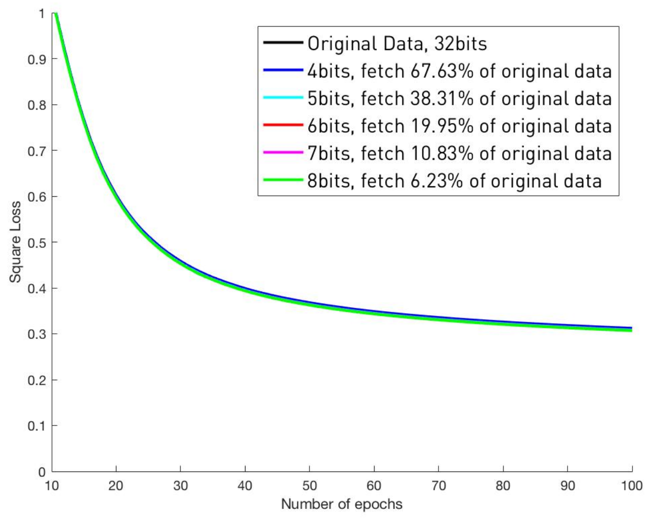


Figure 16: SVM with low precision data and refetching on cod-rna dataset

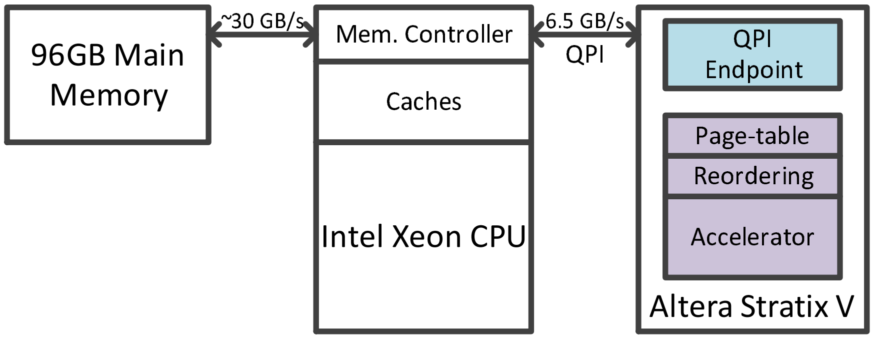


Figure 17: Intel Xeon+FPGA Architecture

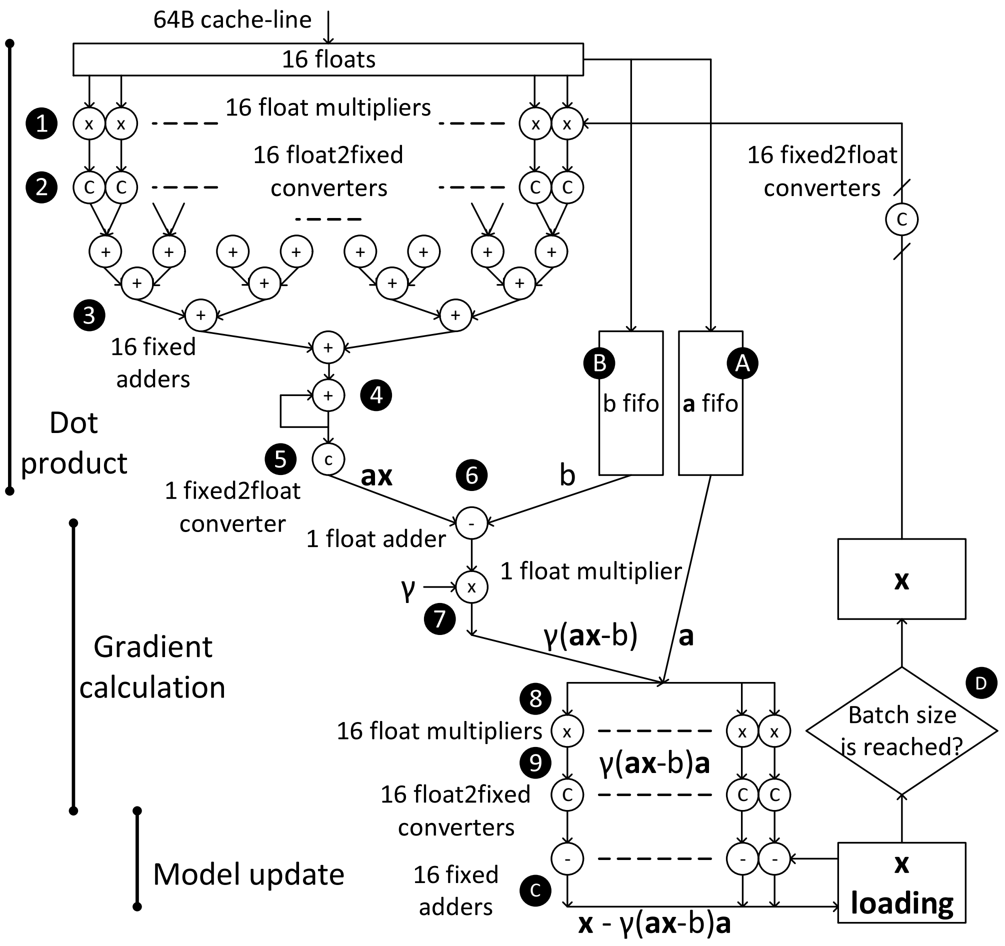
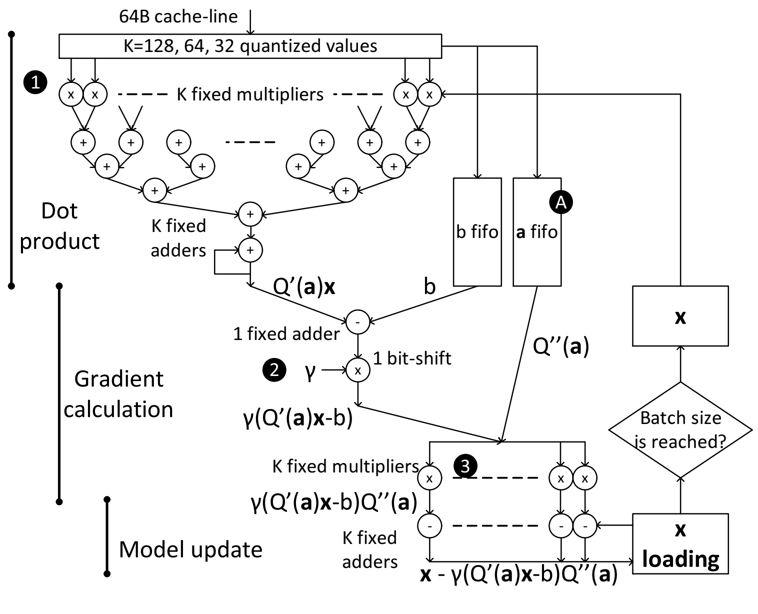
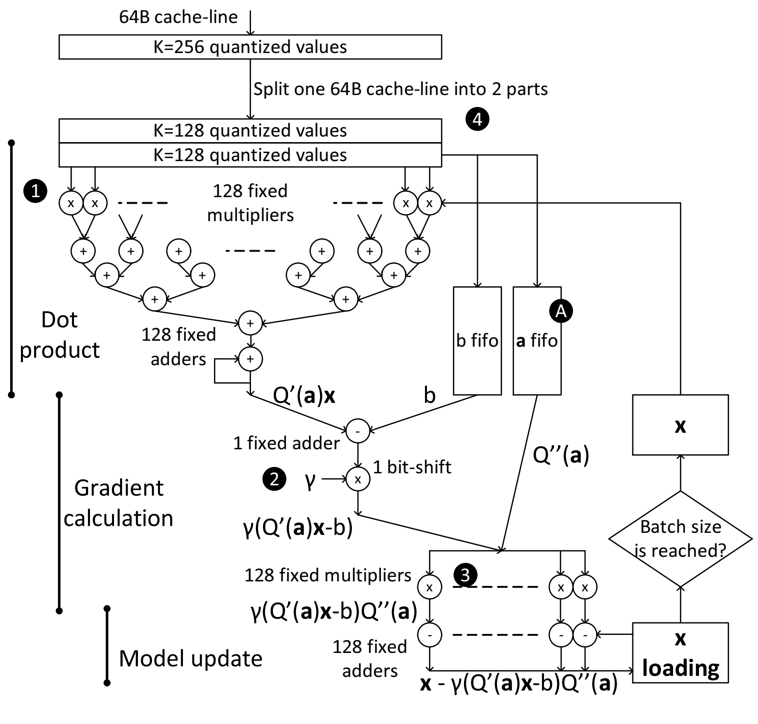


Figure 18: Computation pipeline for *float* FPGA-SGD, with a latency of 36 cycles, a data width of 64B and a processing rate of 64B/cycle.



(a) *Q2*, *Q4* and *Q8* FPGA-SGD, with a latency of $log(K)$+5 cycles, a data width of 64B and a processing rate of 64B/cycle.



(b) *Q1* FPGA-SGD, with a latency of 12 cycles, a data width of 32B and a processing rate of 32B/cycle.

Figure 19: Computation pipelines for all quantizations. Although for *Q2*, *Q4* and *Q8* the pipeline width scales out and maintains 64B width, for *Q1* it does not scale out and the pipeline width needs to be halved, making *Q1* FPGA-SGD compute bound.