JMLR: Workshop and Conference Proceedings 36:46–60, 2014 BIGMINE 2014

Reducing Data Loading Bottleneck with Coarse Feature Vectors for Large Scale Learning

Shingo Takamatsu SHINGO.TAKAMATSU@JP.SONY.COM
Sony Corporation, 2-10-1 Osaki Shinagawa-ku, Tokyo, 141-8610, Japan

Carlos Guestrin GUESTRIN@CS.WASHINGTON.EDU
University of Washington, 185 NE Stevens Way, Seattle, WA 98195, USA

Editors: Wei Fan, Albert Bifet, Qiang Yang and Philip Yu

Abstract

In large scale learning, disk I/O for data loading is often the runtime bottleneck. We propose a lossy data compression scheme with a fast decompression to reduce disk I/O, allocating fewer than the standard 32 bits for each real value in the data set. We theoretically show that the estimation error induced by the loss in compression decreases exponentially with the number of the bits used per value. Our experiments show the proposed method achieves excellent performance with a small number of bits and substantial speedups during training.
Keywords: large scale learning, data compression, error bound analysis

1. Introduction

In large scale learning, the data set often does not fit in the memory of a single machine. Sequentially reading a data set from HDD is commonly used, but disk I/O becomes the bottleneck, not CPU usage (Lin and Kolcz (2012); Golovin et al. (2013); Zhang et al. (2012)). While distributed learning can mitigate the bottleneck, distributed systems are generally hard to implement. Algorithms may need to be modified for the distributed setting, and significant attention must be given to cluster management, fault tolerance, and often unpredictable performance (Kyrola et al. (2012)).

A compact representation of data set reduces the data loading time from HDD. The compact data set can also enable us to store a whole data set in memory. Although this approach needs an initial additional cost of compression, the cost is relatively small since we usually use the same data set multiple times for parameter tuning, model selection, or multiple tasks. We can not use standard data compression tools for these purposes. First, the tools do not always compress machine learning data sets efficiently due to the data's inherently random nature. Second, the tools decompress slowly, often requiring more time than that saved during loading.

We propose a simple data compression scheme for real-valued feature vectors with a fast decompression. While some large data sets take binary values, we have to deal with real values when we employ weighting methods such as tfidf, which can boost the prediction performance (Kogan et al. (2009)). Standard implementations store real values in single precision floating-point representation, using 32 bits per value. This provides a minimal

© 2014 S. Takamatsu & C. Guestrin.

rounding error when running algorithms, but it has a resolution that exceeds the needs of practical machine learning as shown in Section 5.

We use a coarse representation assigning a small number of bits to each real value. The representation has less precision, allowing values to be stored with less memory. Since this approach is simple and does not require a complicated data structure, its decompression is much faster than that of standard data compression tools. The limited resolution loses some information from the original data set, resulting in an additional estimation error called *compression-induced error*. We give a bound for the compression-induced error, showing that the error quickly decreases with the number of bits we use. Our theoretical analysis suggests that in high-dimensional problems our method performs better than subsampling, which is a common technique for reducing the size of a data set by using a subset of the data set.

The contributions of this paper are as follows:

- We propose a lossy data compression scheme with fast decomposition for real-valued data sets that can reduce data loading time without degrading performance. (See Section 3)
- We derive a bound of the compression-induced error in a standard loss minimization problem. The bound decreases exponentially with the number of bits assigned to real values. These results apply to any learning algorithm. (See Section 4)
- The experimental results show that our method achieves almost the same generalization performance with a small number of bits (e.g., 8 bits) as with the original data set, and performs better than subsampling with the same data size. We also show that our data compression can speed up training with sequential data loading from HDD. (See Section 5)

2. Related Work

Vowpal Wabit (VW)[1], which is a machine learning tool for large scale data sets, also has a function to compress feature vectors to make data loading faster. A sparse vector is represented as a set of (feature index, value). VW compresses the feature indices of a sparse feature vector, but do nothing for the values, while the proposed method is for the values. The details of the VW method are described in Section 3. Other commonly used data compression tools, Protocol Buffers[2] and Apache Avro[3], also perform data compression for integers with Variable Byte Code, which assigns a small number of bytes to a small integer, but do not compress real values.

The most closely related technique is presented by Golovin et al. (2013). They reduced the memory needed for the weight vector of online learning by projecting real values in a 32-bit floating-point format onto a coarse discrete set by using randomized rounding. They showed that their method achieves regret guarantees similar to their exact variants

1. http://hunch.net/~vw/
2. https://code.google.com/p/protobuf/
3. http://avro.apache.org/

by using the fact that the errors on gradients caused by the randomized rounding are zero-mean, and as a result, the accumulated error during online learning becomes small. In this work, we reduce the memory needed to store feature vectors, not weight vectors. Similarly to Golovin et al. (2013), we project feature vectors onto a coarse discrete set, but the resultant noise added to the gradient in online updating is no longer zero-mean, which prevents us from using their analysis. We also introduce a scale value for each feature vector, which Golovin et al. (2013) did not use, to have a better resolution with a small number of bits, as written in Section 3.

In this paragraph we discuss other techniques to reduce data set size. Data projection including PCA and random projection can reduce the size of a data set by projecting the feature vectors into a low-dimensional space via a matrix, approximately preserving distances (Bingham and Mannila (2001)). However, the learning performance with data projection can be poor (Shi et al. (2009)). Feature hashing (Weinberger et al. (2009); Shi et al. (2009)) is also commonly used to reduce the number of dimensions of sparse feature vectors, but the data size reduction by this method is moderate since the number of dimensions should not be reduced substantially to avoid the collisions among the non-zero features. Both the approaches project original feature vectors into a low dimensional space, as a result, the resultant feature vectors are hard to interpret. Our method does not suffer from this issue since ours does not change the dimensions. These approaches are orthogonal to ours and can be used with the proposed method at the same time. Li et al. (2011) proposed a compact representation using Minhash for binary feature vectors, while our method is for real-valued feature vectors.

Loss minimization with errors during learning has been widely studied. While most studies considered zero-mean noise or reducing errors on gradients (Bertsekas and Tsitsiklis (2000); Duchi et al. (2012)), in this work, error is added to feature vectors, so we can not directly use the results of those approaches. Our analysis is related to that of Langford et al. (2009), where the authors proposed and analyzed truncated gradient with non-zero-mean and non-reducing errors on gradients in a loss minimization problem. While they focused on regret bounds for online learning, we give error bounds that are not limited in online learning.

3. Rounding for Feature Vector Compression

In this section we focus on positive feature vector $x \in \mathbb{R}_+^p$ for simplicity. $\mathbb{R}_+$ indicates positive real value and p is the number of dimensions of the feature vectors. We can deal with real values including negative ones by taking the absolute value and using one bit for the sign.

A standard fixed-point number format uses a fixed number of bits for the integral part and another fixed number of bits for the fractional part. To deal with feature vectors, the number of bits required by the format tends to be large because the format uses one resolution to cover all the values in the data set. As a result, the assigned bits are not used effectively.[4] To avoid this inefficiency, we use per-feature-vector resolution to leverage the bits as much as possible. A standard floating-point number format covers a larger value

4. For example, the feature vector (2.1, 0.73) needs two bits for the integral part, while (0.9, 0.61) does not use the bits.

Algorithm 1 Scaled Rounding

Input of Compression
A p-dimensional real-valued vector $x = (x_1, x_2, \ldots, x_p)$
The number of bits b

Output of Compression
A p-dimensional b-bit vector $u = (u_1, u_2, \ldots, u_p)$
A scale factor $s \in \mathbb{R}$

Compression
$s \leftarrow x_{max}/N$, where $x_{max} = \max_i x_i$ and $N = 2^b - 1$
for $i = 1$ to p **do**
  $u_i = Round(x_i/s)$, where $Round(\cdot)$ stands for deterministic rounding
end for
Return u and s

Decompression
Return $s \times u = s(u_1, u_2, \ldots, u_p)$

range than the fixed-point one with the same number of bits, while the floating-point one can have larger rounding errors than the fixed-point one in some cases. [5] The proposed method always has smaller rounding errors than the fixed-point number format.

The proposed compression method, named *Scaled Rounding*, is shown in Algorithm 1, which expresses x as the product of a p-dimensional b-bit vector u and a real scale value s.[6] The algorithm defines an equal-interval grid of 2^b points between 0 and the maximum value in x. This per-feature-vector resolution has a more fine-grained resolution for each feature vector than using a global resolution such as the fixed-point format mentioned above. We store b-bit vector u and real value s instead of 32-bit vector x. This significantly reduces the memory to store feature vectors. For instance, if we set b to 4, we can expect to compress the size into roughly 1/8. The decompression is fast because it is just a multiplication of a real value to an integer vector. We employ deterministic rounding rather than randomized one to have a smaller L2 distance from the original vector.

When we compress a sparse vector, we express the vector as the sequence of feature indices and the sequence of values, and compress the two sequences separately. The size of the sequence of indices is also large because standard implementations use 32 bits to store a integer. As mentioned above, the machine learning tool, VW, has a function to compresses feature indices. The VW method is formally written in Algorithm 2. The VW method compresses the sequence of feature indices by applying Variable Byte Code to the gaps between the indices.[7] The gaps between integers in an integer sequence tend to be small,

5. For example, the worst rounding error of a four-bit floating-point number format with three bits of exponent and one bit of fraction is 2.0.
6. For example, in the two-bit setting the vector (0.9, 0.61) is represented as the scale value 0.3 and the two-bit vector (11, 10).
7. For simplicity, we assume feature indices are in ascending order.

Algorithm 2 The VW method

Input
A sequence of p feature indices $z = (z_1, z_2, \ldots, z_p)$ where the elements are sorted in ascending order

Compression
$ByteArray \leftarrow \{\}$.
Append $VariableByteCode(z_1)$ to $ByteArray$
for $i = 2$ to p **do**
  Append $VariableByteCode(z_i - z_{i-1})$ to $ByteArray$
end for
Return $ByteArray$

and this approach is traditionally used in information retrieval to compress an index file (Manning et al. (2008)). Given sparse feature vectors, we use Scaled Rounding to compress the values, and employ the VW method to compress the feature indices.

4. Theoretical Analysis on Loss Minimization

We train parameters with the feature vectors compressed by Scaled Rounding. This results in different parameters from those estimated with the original feature vectors. In this section, we prove a bound of the compression-induced error caused by the loss with the compression. Then by using the bound, we give a condition where Scaled Rounding works better than subsampling.

We consider minimizing a function

$$F(w; X, Y) = \frac{1}{n} \sum_{i=1}^{n} f(w; x_i, y_i).$$

Here $(X, Y) = \{(x_i, y_i) | x_i \in \mathcal{X}, y_i \in \mathcal{Y}, i = 1, \ldots, n\}$[8], n is the number of input-output pairs, and $f(w; \cdot, \cdot) : \mathcal{W} \subset \mathbb{R}^p \to \mathbb{R}$ is a convex loss function that satisfies Assumptions 1. We also assume the convex set $\mathcal{W}$ contains the optimal point. We use $f(w)$ instead of $f(w; x, y)$ when we do not need to specify x or y. In this paper norm indicates L2 norm if not specified.

Assumptions 1 *The loss function f satisfies the following assumptions:*

1. *f is strongly convex on w, i.e., there is a constant c for all w' and $w \in \mathcal{W}$ such that*

$$f(w') \geq f(w) + (w' - w)^T \nabla f(w) + \frac{1}{2} c \, \| w' - w \|^2.$$

2. *∇f is Lipschitz continuous on w and x, i.e., there exist constants L_w and L_x such that*

$$\| \nabla f(w'; x, y) - \nabla f(w; x, y) \| \leq L_w \| w' - w \| \quad for \ \forall (w', x, y), \forall (w, x, y) \in (\mathcal{W} \times \mathcal{X} \times \mathcal{Y}),$$
$$\| \nabla f(w; x', y) - \nabla f(w; x, y) \| \leq L_x \| x' - x \| \quad for \ \forall (w, x', y), \forall (w, x, y) \in (\mathcal{W} \times \mathcal{X} \times \mathcal{Y}).$$

8. x_i is a vector from this section.

We use $L = \max(L_w, L_x)$ *in this analysis.*

The second assumption is natural if we consider $f(g(w^T x), y)$ with a link function $g : \mathbb{R} \to \mathbb{R}$. This type of loss functions includes logistic loss and squared loss.

In this analysis, the compressed feature vector is written as the following noisy vector:

$$\hat{x}_{ij} = x_{ij} + \epsilon_{ij}, \text{ where } \|\epsilon_{ij}\| \leq B, \ j = 1, \ldots, d_i. \tag{1}$$

Here j denotes index of nonzero elements of the i-th feature vector[9] and d_i is the number of non-zero elements of x_i. B is the norm bound of the noise for all i and j. When we employ Scaled Rounding, $B = R/2(2^b - 1)$, where R is a constant such that $\| x_i \|_\infty \leq R$ for $i = 1, \ldots, n$. The compression-induced error is defined as $F(\hat{w}^*; X, Y) - F(w^*; X, Y)$, where $w^* = \text{argmin}_w F(w; X, Y)$ and $\hat{w}^* = \text{argmin}_w F(w; \hat{X}, Y)$. Here, $\hat{X} = \{(\hat{x}_i) | i = 1, \ldots, n\}$.

4.1. Compression-Induced Error Bound

We start by considering stochastic gradient descent (SGD) while our main result, or Theorem 4, is not restricted to SGD. We update parameter vector w with noisy vector $\hat{x}$ instead of x by SGD,

$$w_{t+1} = w_t - \eta_t \nabla f(w_t; \hat{x}_t, y_t), \tag{2}$$

where η_t is the stepsize at time t and $\nabla f(w_t; \hat{x}_t, y_t)$ is subgradient with $(\hat{x}_t, y_t)$ randomly sampled at t. Here, $\hat{x}_t$ is from Eq.(1). Our analysis extends the technique of Nomirovski et. al. (2009).

Lemma 2 *Suppose we use the update rule Eq.(2) and* $\| \nabla f(w) \| \leq M$ *with some constant* M. *Let* $d = \max_i d_i$. *Then we have*

$$\| w_{t+1} - w^* \|^2 \leq (1 - 2\eta_t c) \| w_t - w^* \|^2 + \eta_t^2 \left(M^2 + 2MLB\sqrt{d} \right) + 2\eta_t LB\sqrt{d} \| w_t - w^* \| .$$

The proofs of this lemma and the next lemma are shown in Appendix. The rightmost term is from the noise added to the feature vector. Lemma 2 shows that, even with the noisy feature vectors, SGD can still improve the objective.

Lemma 3 *Suppose* $\beta = \max_{t' \leq \infty} \| w_{t'} - w^* \|$ *and take stepsizes* $\eta_t = \theta/t$ *for some constant* $\theta > 1/c$, *then we have*

$$\| w_t - w^* \|^2 \leq \frac{\theta^2 \left(M^2 + 2MLB\sqrt{d} \right)}{t(2\theta c - 1)} + 2\theta LB\sqrt{d}\beta.$$

9. j should be written as j_i. For simplicity we omit the subscript.

The main theorem shows that the noisy feature vectors do not significantly affect the optimum parameters or objective value. The theorem holds for any learning algorithm not just SGD.

Theorem 4 *Suppose we use $\hat{X}$ made by Scaled Rounding in Algorithm 1, then we have*

$$\begin{aligned} \| \hat{w}^* - w^* \|^2 &\leq \frac{\theta L R \sqrt{d} \beta}{2^b - 1} \quad \text{and} \\ F(\hat{w}^*; X, Y) - F(w^*; X, Y) &\leq \frac{\theta L^2 R \sqrt{d} \beta}{2(2^b - 1)}. \end{aligned}$$

Proof By taking $t \to \infty$ in Lemma 3, we obtain the first inequality because the SGD with $\hat{X}$ converges to $\hat{w}^*$ with an infinite number of iterations. The second inequality in the theorem follows from the Lipschitz continuous on w, i.e.,

$$\begin{aligned} &\| \nabla f(w; x, y) - \nabla f(w^*; x, y) \| \leq L \| w - w^* \| \\ \Rightarrow \quad &f(w; x, y) \leq f(w^*; x, y) + \frac{1}{2} L \| w - w^* \|^2 . \end{aligned}$$

■

Theorem 4 shows that the compression-induced error bound gets exponentially smaller as one adds bits b, and becomes zero when $b \to \infty$. We can control the tradeoff between compactness and accuracy with the number of the bits. The bounds do not depend on the number of feature vectors. This implies that we can use the proposed method safely with a large number of feature vectors.

When we assume the loss function is linearly parameterized by w, i.e. $f(g(w^T x), y)$ with a link function $g : \mathbb{R} \to \mathbb{R}$, Theorem 4 can be rewritten with a smaller constant especially for sparse feature vectors since the noises on a sparse feature vector x_i affect only d_i elements of the parameter in updating. The proof of the next corollary is shown in Appendix.

Corollary 5 *In addition to the assumptions of Theorem 4, suppose $f(g(w^T x), y)$ with a link function g, and let $\gamma = \max_{t' \leq \infty} \| w_{t'} - w^* \|_\infty$. Then we have*

$$\| \hat{w}^* - w^* \|^2 \leq \frac{\theta L R d \gamma}{2^b - 1}.$$

4.2. Comparison with Subsampling

Scaled Rounding adds noise to feature vectors to reduce the size of the data set, and as the result, we have the compression-induced error after learning. A commonly used approach for the same purpose is the subsampling method that uses a subset of the data set to reduce the size. By using the bounds in this section, we compare the two methods theoretically and derive a condition under which Scaled Rounding works better than subsampling given the same memory size. The empirical comparison to subsampling is shown in the next section.

Table 1: Data set properties. The number of training/testing feature vectors and the number of features, as well as the sizes of training data sets in binary format.

	task	# training	# testing	# features	data size
Webspam	classification	280,000	70,000	680,715	8.2G
RCV1	classification	677,399	20,242	47,236	392M
AdClick	CTR prediction	8,000,000	2,000,000	223,021	1.9G
10K	regression	16,087	3,308	150,360	156M

In this analysis we follow Bottou and Bousquet (2008), and assume that the loss function is linearly parameterized by w. We assume the same model and the same learning algorithm, so we do not consider the approximation error and the optimization error. With the data compression the learning error can be decomposed into the two parts: the standard estimation error caused by a finite number of data, and the compression-induced error. Following Bottou and Bousquet (2008), a bound of the estimation error is given by $A\sqrt{p/n}$ with some constant A, where n and p are the number of feature vectors and the dimension of parameters, respectively. We use Corollary 5 to bound the compression-induced error. For subsampling the compression-induced error is zero, but the number of feature vectors is limited to $m < n$. The desired condition in terms of bounds can be written as follows:

$$A\sqrt{\frac{p}{m}} > A\sqrt{\frac{p}{n}} + \frac{A'd}{2^b} \;\Rightarrow\; 2^b > \frac{A'd}{A\left(\sqrt{\frac{p}{m}} - \sqrt{\frac{p}{n}}\right)}. \tag{3}$$

Here A' is constant from Corollary 5.

To understand Eq.(3) we assume the compression ratio with Scaled Rounding to be 25%, then let $m = n/4$ since we assume the same memory size. The right side of Eq.(3) becomes $\frac{A'd}{A}\sqrt{\frac{n}{p}}$. We can see that Scaled Rounding works better than the subsampling method when the problem has a large p relative to n and a small d, or sparse features.

5. Experiments

We evaluated Scaled Rounding on a variety of machine learning tasks with public data sets shown in Table 1. Webspam and RCV1 are for classification, and AdClick is for Click Through Rate (CTR) prediction, and 10K is for regression. Webspam, RCV1, and 10K were from Chang and Lin (2011). The AdClick data set was from KDD Cup 2012[10]. In all data sets, we computed tfidf for token features and had positive real-valued feature vectors. In advance the data sets were converted from text format to binary format, which uses 32 bits for a feature index or a value. This dramatically reduces parsing time to construct feature vectors. Logistic regression was employed for classification and CTR prediction, and ridge regression was used for regression. The evaluation metrics are accuracy for classification and Root Mean Squared Error (RMSE) for CTR prediction and regression. The training data sets were used for parameter training with SGD and hyperparameter tuning. The

10. http://www.kddcup2012.org/c/kddcup2012-track2

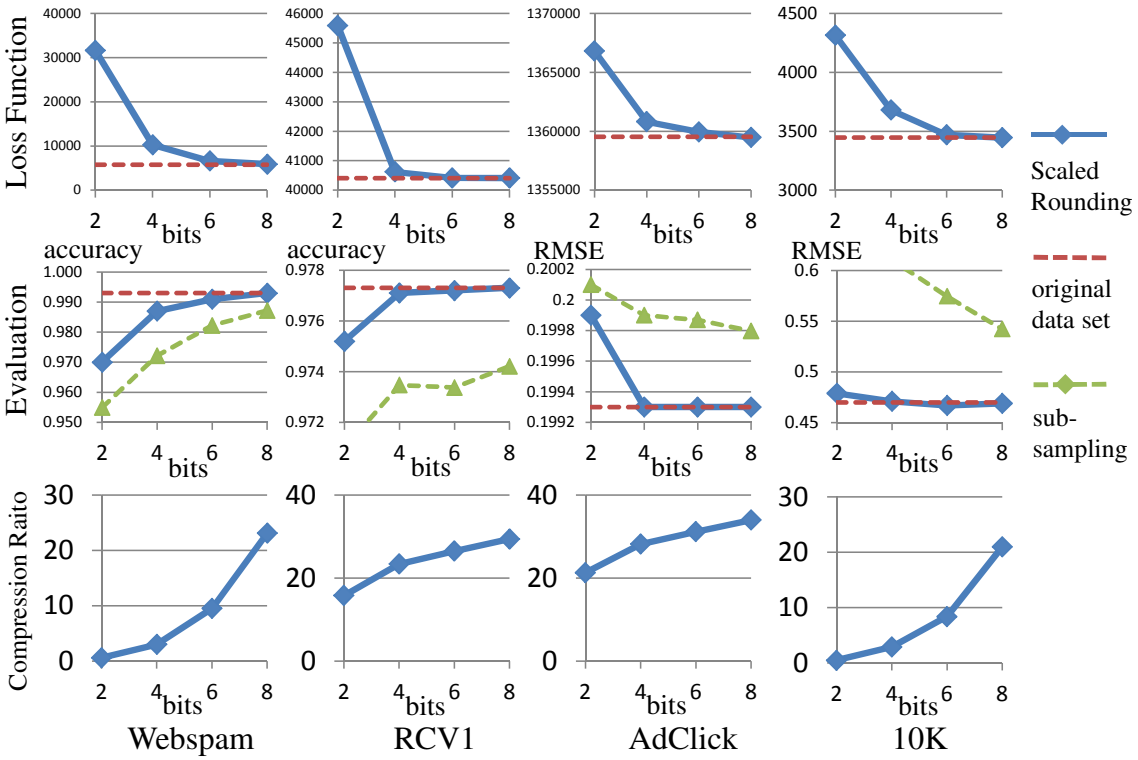

Figure 1: Learning performance. The horizontal axes indicate the number of bits we used. The top row shows the values of loss functions after 30 epochs of SGD. The middle row shows evaluation performance. The bottom row shows compression ratio. The blue solid lines indicate Scaled Rounding, the red dashed lines indicate the original data set, and the green dashed lines indicate subsampling.

testing data sets were used for evaluation. The implementations were coded in Java and run on an identical hardware configuration, where the processor was Xeon (3.00GHz).

We first report the learning performance with Scaled Rounding compared with that using the original data sets and that with subsampling. Next we report detailed performance of compression with Scaled Rounding and compare with VW and a fast data compression tool, LZ4[11], which a modern data compression tool focusing on compression/decompression speed.

5.1. Learning Performance

We ran SGD with the compressed data set and report the values of loss function after training as well as prediction performance. The values are compared with those with the original data sets. We also compared with subsampling. At each number of bits we plotted

11. We used the implementation from https://github.com/jpountz/lz4-java. We compared LZ4 with Snappy(https://code.google.com/p/snappy/), but LZ4 had a better performance.

the prediction performance of subsampling with the same sampling ratio as the compression ratio of Scaled Rounding. The results of subsampling were averaged over five samplings.

The results are shown in Figure 1. We see that in all the tasks the compression with 8 bits achieves almost same training loss and prediction performance as those with the original data sets. The compression ratios are also shown in Figure 1. A tradeoff can be observed between the value of loss function and the compression ratio, dominated by the number of the bits. In this experiment, when the value of the loss function was small, the setting had a good prediction performance. This implies that we can choose an appropriate number of bits by looking at the value of loss function during training. In RCV1 and AdClick, p is not big compared to n, so the theoretical analysis in Section 4.2 suggests the subsampling method would work better than the proposed method. Nevertheless, we see that Scaled Rounding worked better than subsampling in all the data sets including RCV1 and AdClick.

5.2. Compression Performance

We used the training data sets of Webspam and AdClick in this experiment. Assuming the data sets do not fit in memory, we consider the following steps for each iteration of SGD: 1. load the byte array corresponding to a feature vector to the memory, 2. decompress (if needed) and parse the byte array to have the feature vector, 3. update the parameter with the feature vector. In our experiment the first step took longer than the time to process the second and third steps under the 165MB/sec data loading throughput, which was the average throughput in our implementation. In this situation, we use multi-threading and have one thread for each step. The total processing time of this learning system is determined by the first step when the second and third steps are faster than the first one. Since the learning steps can be made faster by using the parallel SGD (Recht et al. (2011)), we focus on the first and second steps. In this experiment we report the compression ratio, compression time, and time needed in the second step, as well as the data loading time with the 165MB/sec throughput.

In the binary format each sparse feature vector was stored as the sequence of feature indices and the sequence of values. To measure a pure performance of Scaled Rounding, which is for real values, we separated the original binary file into a value-only file and an index-only file.[12] The value-only file is used to measure a pure performance of Scaled Rounding. The value-only files roughly correspond to data sets with dense feature vectors since such data sets do not store feature indices. The index-only files correspond to binary feature data sets.

The results with the value-only files are shown in Table 2. We compared the following methods: (a) that using binary format as the baseline, (b) Scaled Rounding with 8 bits, and (c) LZ4. In fact, standard data compression tools like gzip did not help to speed up the learning system. In our experiment the decompression of gzip took substantially longer than the data loading. Table 2 shows that Scaled Rounding achieves 5.4 speedup in loading Webspam, and 3.3 speedup with AdClick. We can also see that the decompression of Scaled Rounding did not take longer than the loading time. The compression ratio of Webspam

12. For example, the sparse vector, (1:2.1, 12:0.72), is separated into the value part, (2.1, 0.72), and the index part, (1, 12).

Table 2: Compression ratio, compression time, decompression and parsing time, and estimated data loading time, which is (data size / 165MB). The upper half shows the results with the value-only files, the lower half shows those with the original files.

		comp. ratio	comp. time	decomp. and parse time	estimated loading time
Webspam (value-only)	(a) binary format	100%	N/A	2.9 sec	25.3 sec
	(b) Scaled Rounding	18.5%	12.9 sec	1.0 sec	4.7 sec
	(c) LZ4	99.9%	3.8 sec	2.8 sec	25.0 sec
AdClick (value-only)	(a) binary format	100%	N/A	1.3 sec	5.9 sec
	(b) Scaled Rounding	30.7%	3.6 sec	1.0 sec	1.8 sec
	(c) LZ4	57.0%	4.9 sec	2.9 sec	3.4 sec
Webspam	(A) binary format	100%	N/A	5.4 sec	50.6 sec
	(B) VW	67.4%	16.6 sec	9.9 sec	34.1 sec
	(C) Scaled Rounding + VW	22.6%	22.2 sec	6.0 sec	11.5 sec
	(D) LZ4	98.2%	14.1 sec	8.7 sec	49.7 sec
AdClick	(A) binary format	100%	N/A	1.7 sec	11.8 sec
	(B) VW	70.5%	4.5 sec	3.3 sec	8.3 sec
	(C) Scaled Rounding + VW	35.9%	7.3 sec	2.9 sec	4.2 sec
	(D) LZ4	54.5%	9.5 sec	5.2 sec	6.4 sec

was smaller than that of AdClick. This is because a feature vector in the Webspam data set has relatively small values that were rounded to zeroes and such features were discarded. This effect also helps to reduce the decompression time of Scaled Rounding. LZ4 did not successfully compress the Webspam data set. On the other hand, LZ4 compressed AdClick to 55.4% because AdClick has features commonly appearing in the feature vectors. The compression speed by Scaled Rounding was competitive to that by LZ4.

Since the results with the index-only files do not directly relate to the proposed method, we do not report the detailed results. Instead, we summarize the results. The VW method loaded the Webspam/AdClick data sets 2.9/2.6 times faster than the binary format. The VW method was better than that applying Variable Byte Code directly to feature indices. Reassigning feature indices so that more frequent features have smaller indices did not improve the VW method because this index reassignment did not always reduce the gaps between neighbor feature indices.

We finally report the compression performance with the original files containing both values and indices. We compared the following methods: (A) that using the binary format as the baseline, (B) VW's data compression, (C) the method using Scaled Rounding for values and the VW method for indices, and (D) LZ4. The results are shown in Table 2. The results show that the method with Scaled Rounding achieves 4.4/2.8 speedups to load the Webspam/AdClick data sets. The decompression by Scaled Rounding was fast enough to hide its computation behind the data loading. The results also suggest that by using the proposed method we can make the learning system substantially faster than VW. LZ4 slightly improved the data loading time with AdClick, but did not work for Webspam, while the proposed method worked well in both the data sets.

6. Conclusions

In this paper, we theoretically and empirically show that feature vectors in a coarse finite precision can substantially reduce the disk I/O bottleneck in loading real-valued data sets without degrading predictive performance. Our theory suggests that our method works especially well for high-dimensional sparse problems, which have a variety of applications including text classification and recommendation systems. The proposed method can be useful to other usage such as helping a large data set to fit in memory, or reducing data communication time in distributed systems.

Acknowledgments

We thank Tyler Johnson, Joseph Bradley, and the anonymous reviewers for feedback and helpful discussions.

References

Dimitri P. Bertsekas and John N. Tsitsiklis. Gradient convergence in gradient methods with errors. *SIAM Journal on Optimization*, 10(3):627–642, 2000.

Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 245–250, 2001.

Leon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems 20*, pages 161–168, 2008.

Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, 2011.

John Duchi, Michael Jordan, and Martin Wainwright. Privacy aware learning. In *Advances in Neural Information Processing Systems 25*, pages 1439–1447. 2012.

D. Golovin, D. Sculley, H. Brendan McMahan, and M. Young. Large-scale learning with less ram via randomization. In *Proceedings of the 30th Intenatinal Conference on Machine Learning*, pages 325–333, 2013.

Shimon Kogan, Dimitry Levin, Bryan R. Routledge, Jacob S. Sagi, and Noah A. Smith. Predicting risk from financial reports with regression. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 272–280, 2009.

Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. Graphchi: Large-scale graph computation on just a pc. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation*, pages 31–46, 2012.

John Langford, Lihong Li, and Tong Zhang. Sparse online learning via truncated gradient. *The Journal of Machine Learning Research*, 10:777–801, 2009.

Ping Li, Anshumali Shrivastava, Joshua L. Moore, and Arnd C. König. Hashing algorithms for large-scale learning. In *Advances in Neural Information Processing Systems 24*, pages 2672–2680, 2011.

Jimmy Lin and Alek Kolcz. Large-scale machine learning at twitter. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 793–804, 2012.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. 2008.

A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems 24*, pages 693–701, 2011.

Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alexander J. Smola, Alexander L. Strehl, and Vishy Vishwanathan. Hash kernels. *Journal of Machine Learning Research - Proceedings Track*, 5:496–503, 2009.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120, 2009.

Caoxie Zhang, Honglak Lee, and Kang Shin. Efficient distributed linear classification algorithms via the alternating direction method of multipliers. *Journal of Machine Learning Research - Proceedings Track*, pages 1398–1406, 2012.

Appendix

Proof of Lemma 2 Consider the parameter vector updated without the noise, i.e., $\tilde{w}_{t+1} = w_t - \eta_t \nabla f(w_t; x_t, y_t)$.

$$
\begin{aligned}
&\| w_{t+1} - w^* \|^2 \\
\leq &\| w_{t+1} - w^* \|^2 + \| w_{t+1} - \tilde{w}_{t+1} \|^2 \\
= &\| \tilde{w}_{t+1} - w^* \|^2 - 2(w^* - w_{t+1})^T (w_{t+1} - \tilde{w}_{t+1}) \\
= &\| \tilde{w}_{t+1} - w^* \|^2 + 2\eta_t (w^* - w_{t+1})^T (\nabla f(w_t, \hat{x}_t, y_t) - \nabla f(w_t, x_t, y_t)) \\
\leq &\| \tilde{w}_{t+1} - w^* \|^2 + 2\eta_t LB\sqrt{d} (\| w_t - w^* \| + \eta_t M) \qquad \text{(See below)} \\
= &\| w_t - w^* \|^2 + \| \tilde{w}_{t+1} - w_t \|^2 - 2(w_t - w^*)^T (w_t - \tilde{w}_{t+1}) + 2\eta_t LB\sqrt{d} (\| w_t - w^* \| + \eta_t M) \\
= &\| w_t - w^* \|^2 + \eta_t^2 \| \nabla f(w_t, x_t, y_t) \|^2 - 2\eta_t (w_t - w^*)^T \nabla f(w_t, x_t, y_t) + 2\eta_t LB\sqrt{d} (\| w_t - w^* \| + \eta_t M) \\
\leq &(1 - 2\eta_t c) \| w_t - w^* \|^2 + \eta_t^2 M^2 + 2\eta_t LB\sqrt{d} (\| w_t - w^* \| + \eta_t M) \\
= &(1 - 2\eta_t c) \| w_t - w^* \|^2 + \eta_t^2 \left(M^2 + 2MLB\sqrt{d} \right) + 2\eta_t LB\sqrt{d} \| w_t - w^* \| .
\end{aligned}
$$

The last inequality follows from $(w-w^*)^T\nabla f(w;x,y) \geq c \parallel w-w^* \parallel^2$ derived from the strong convexity and $\nabla f(w^*;x,y)=0$. The second inequality follows from

$$\begin{aligned}
&(w^*-w_{t+1})^T(\nabla f(w_t,\hat{x}_t,y_t)-\nabla f(w_t,x_t,y_t))\\
&=(w^*-w_t+\eta_t\nabla f(w_t;\hat{x}_t,y_t))^T(\nabla f(w_t,\hat{x}_t,y_t)-\nabla f(w_t,\tilde{x}_t,y_t))\\
&\leq(\parallel w_t-w^*\parallel+\eta_t\parallel\nabla f(w_t;\hat{x}_t,y_t))\parallel)\parallel\nabla f(w_t,\hat{x}_t,y_t)-\nabla f(w_t,\tilde{x}_t,y_t)\parallel\\
&\leq(\parallel w_t-w^*\parallel+\eta_t\parallel\nabla f(w_t;\hat{x}_t,y_t))\parallel)L\parallel\epsilon_t\parallel\\
&\leq(\parallel w_t-w^*\parallel+\eta_t\parallel\nabla f(w_t;\hat{x}_t,y_t))\parallel)LB\sqrt{d}\\
&\leq LB\sqrt{d}(\parallel w_t-w^*\parallel+\eta_t M).
\end{aligned}$$

Here ϵ_t is the noise vector added to x_t and d is the maximum number of non-zero elements in an input vector in the data set. The first inequality follows from the Cauchy-Schwarz inequality, the second from L-Lipschitz on x, the third from Eq.(1) and the last from the assumption on M. ■

Proof of Lemma 3 The lemma follows from induction. With $t=1$ we have

$$\begin{aligned}
&\parallel w_2-w^*\parallel^2\leq(1-2\theta c)\parallel w_1-w^*\parallel^2+\theta^2\left(M^2+2MLB\sqrt{d}\right)+2\theta LB\sqrt{d}\beta\\
\Leftrightarrow\ &\parallel w_1-w^*\parallel^2\leq\frac{\theta^2\left(M^2+2MLB\sqrt{d}\right)}{2\theta c-1}+2\theta LB\sqrt{d}\beta.
\end{aligned}$$

The last inequality follows from $2\theta c-1>1$. Assuming the lemma holds at t, we have

$$\begin{aligned}
&\parallel w_{t+1}-w^*\parallel^2\\
&\leq\left(1-\frac{2\theta c}{t}\right)\parallel w_t-w^*\parallel^2+\frac{\theta^2}{t^2}\left(M^2+2MLB\sqrt{d}\right)+\frac{2\theta}{t}LB\sqrt{d}\beta\\
&\leq\left(1-\frac{2\theta c}{t}\right)\left(\frac{\theta^2\left(M^2+2MLB\sqrt{d}\right)}{t(2\theta c-1)}+2\theta LB\sqrt{d}\beta\right)\\
&\qquad+\frac{\theta^2}{t^2}\left(M^2+2MLB\sqrt{d}\right)+\frac{2\theta}{t}LB\sqrt{d}\beta\\
&=\left(\frac{1}{t}-\frac{1}{t^2}\right)\frac{\theta^2\left(M^2+2MLB\sqrt{d}\right)}{(2\theta c-1)}-\frac{(2\theta c-1)\,2\theta LB\sqrt{d}\beta}{t}+2\theta LB\sqrt{d}\beta\\
&\leq\frac{\theta^2\left(M^2+2MLB\sqrt{d}\right)}{(t+1)(2\theta c-1)}+2\theta LB\sqrt{d}\beta.
\end{aligned}$$

The last inequality follows from $1/t-1/t^2<1/(t+1)$ for $t>0$ and $2\theta c-1>0$. ■

Proof of Corollary 5 We first prove

$$\|w_{t+1}-w^*\|^2\leq(1-2\eta_t c)\parallel w_t-w^*\parallel^2+\eta_t^2M^2+2\eta_t LBd\gamma.$$

Its proof is almost identical to Lemma 2 except

$$
\begin{aligned}
&(w^* - w_{t+1})^T \left(\nabla f(g(w_t^T x_t), y_t) - \nabla f(g(w_t^T \hat{x}_t) y_t)\right) \\
&= (W_t^*)^T \left(\nabla f(g(w_t^T x_t), y_t) - \nabla f(g(w_t^T \hat{x}_t) y_t)\right) \\
&\leq \| W_t^* \| \| \nabla f(g(w_t^T x_t), y_t) - \nabla f(g(w_t^T \hat{x}_t) y_t) \| \\
&\leq \sqrt{d}\gamma L \| \epsilon_t \| \\
&\leq LBd\gamma,
\end{aligned}
$$

where t-th element of W_t^* is $w^* - w_{t+1}$ when $w_t \neq 0$ and zero otherwise. The first equality holds because the subgradient of $f(g(w^T x), y)$ can be written as $h(w, x, y)x$ with a function $h : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ that depends on f and g. The rest of the proof is almost same as those of Lemma 3 and Theorem 4. ■