SketchML: Accelerating Distributed Machine Learning with Data Sketches

0. Abstract

gradients are sparse and nonuniformly distributed

contributions:

- a sketch based method that compresses the gradient values.

  利用概率数据结构逼近数据分布

  quantile sketch : sort gradient values into buckets and encodes them with the bucket indexes.

- (进一步压缩bucket index) MinMaxSketch builds a set of hash tables and solves hash collisions with a MinMax strategy.
- delta-binary encoding method that calculates the increment of the gradient keys and stores them with fewer bytes.

1. Introduction

1.1 Background and Motivation

数据量太大，无法使用中心化的系统去训练

因此，部署到分布式系统上是必不可免的

the communication often dominates the total cost

减少梯度移动的好处：

Case 1: Large Model

模型越复杂，对于 users or objects 越具有代表性

越具代表性的模型，更有可能有更高质量的 prediction

但是大规模的集群还会带来相当大的通信，这会阻碍整体的性能

因此，squeeze the transferred data 是非常重要的

Case 2: Cloud Environment

对于云计算服务，尽量减少网络传输是一个永恒的目标

Case 3: Geo-Distributed ML

在 data centers 之间 move data 不是切实可行的

在 WAN 上的 data movement 比在 LAN 上要慢很多

Case 4: Internet of Things (IoT)

IoT 尝试在一些物联网设备上收集并交换数据

an efficient communication infrastructure is of great value.

解决上述cases 的办法通常是 compression techniques.

现有的压缩方法可以分为两类：

lossless methods and lossy methods

Lossless methods

用于有重复的整数，eg. Huffman Coding, RLE, DEFLATE and Rice.

但是不能用于非重复的或者是浮点型数

Lossy methods

用于 compress floating-point gradients ,通过基于 truncation 裁剪 或者 quantization 的策略的 threshold.

基于 truncation 的 threshold 过于激进，难以使得 ML算法收敛

quantization approach 是更折中的，实现了 compression 和 convergence 的 tradeoff

但是现有的 quantization approach 有两个不切实际的假设

1. 假设 a gradient vector to be compressed is dense.

   但是由于 training data sparsity, gradient vectors are sparse.

   (1) 压缩一个稀疏梯度向量会浪费大量的时间

   (2) 如果以(key, value) 存储稀疏梯度向量的时候, the gradient keys 不能被压缩
2. 假设 the gradient values 服从 a uniform distribution.

   但是实际中，梯度向量的梯度值服从的是 a nonuniform distribution.

   Worse，大部分梯度值都分布在接近0 的范围内

   uniform quantification 方法不能满足 gradient values 的统计分布

研究问题：

what data structure should we use to compress a sparse gradient vector

现有的 gradient compression 方法是为 dense and uniformly distributed gradient设计的，因此在 sparse and nonuniform distributed setting上效果不佳

SketchML, 支持 sparse gradients 并且满足 gradients 的统计分布

对于$(k_j, v_j)_{j=1}^{d}$的键值对:

- sketch-based algorithm compress values
- delta-binary encoding method to compress keys.

1.2 Overview of Technical Contributions

Data Model

How to Compress Gradient Values

- Quantile sketches 被用来估计 items 的分布
- Frequency sketches 被用来估计 items 发生的频率

提出使用 quantile sketch 把 gradient values 分到几个 buckets，之后通过对应的 bucket index $b(v_j)$ encode each value.

使用 a binary representation to encode the bucket indexes. (减小通信开销)

MinMaxSketch

- encodes the bucket indexes using multiple-hashing approximation.
- MinMax strategy to solve the hash collision problem during insertion phase and query phase.
- a dynamic learning rate schedule to compensate the vanishing of gradients.
- a grouping method to decrease quantification error.

How to Compress Gradient Keys

gradient keys 不同于 gradient values，keys 有更高的准确度要求

需要的是a lossless method to compress gradient keys

因为 key-value pairs 是按照 keys 存储的，意味着 keys 是递增排序，因此提议以 delta format 去存储 key, 即存储临近 keys 的 difference

即使 key 会比较大，但是 neighboring keys difference是在一个 small range.

因此可以用二进制表示来存储 keys

2. Preliminaries

2.1 Definition of Notations

- W: number of workers.
- N: number of training instances.
- D: number of model dimensions.
- g: a gradient vector.
- d: number of nonzero dimensions in a gradient vector.
- (k_j, v_j): j-th nonzero gradient key and gradient value in a sparse gradient vector.
- m: size of a quantile sketch.
- q: number of quantile splits.
- s, t: row and column of MinMaxSketch. s denotes the number of hash tables, and t denotes the number of bins in a hash table.
- r: group number of MinMaxSketch.

2.2 Data Model

our goal: 在把梯度发送出去之前，compress the gradients $\{g^w\}_{w=1}^W$

2.3 Quantile Sketch

analyzing the distribution of item values in a single pass.

brute-force sorting

> computation complexity O(NlogN)
>
> space complexity O(N)

Quantile sketch

使用small data structure 去近似在a single pass 中的 item value 的精确分布

quantile sketch 的主成分是 quantile summary.

两个主要操作： merge and prune.

merge: combine two summaries into a merged summary。

prune: reduce the number of summaries to avoid exceeding the maximal size.

a quantile sketch 有 m 个 quantile summaries.

> computation complexity: O(N)
>
> space complexity: O(m)

比 brute-force 提升了很多

而且已有的 quantile sketches 方法提供了强error bounds

2.4 Frequency Sketch

关注的是 repeated occurrences of items.

frequency sketch 被用来估计 the frequency of different values of items.

Count-min sketch (类似 Bloom Filter)

data structure: s rows and t columns, 用 H 表示

每一行都是 t-bin hash table, 都有一个 hash function $h_i(-)$

insertion phase

- 对于每一行，计算 hash function $h_i(x)$, 来表示 a column index.
- 对 H 中对应的值+1　计数

query phase

- 从 H 中得到 s 个 candidates.、
- 选择 the minimum 作为 final result.

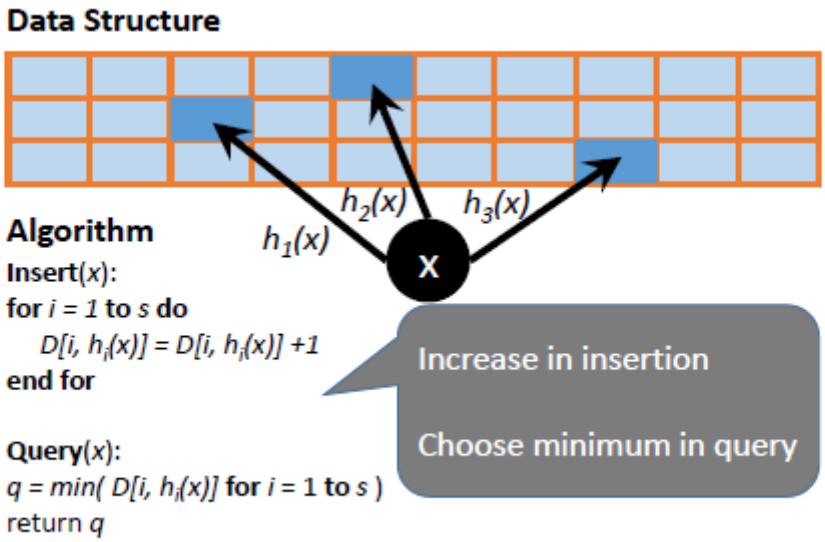


Figure 1: A Frequency Sketch

hash collision

- ignore hash collisions
- increases the hash bin once it is chosen.

3. The Framework of SketchML

3.1 Overview of The Framework

major components:

- quantile-bucket quantification
- MinMaxSketch

  compress gradient values.
- dynamic delta-binary encoding

  compress gradient keys.

Encode Phase

1. quantile sketch, 生成 candidate splits, 利用 bucket sort 去 summarize the values.
2. values 用对应的 bucket indexes 表示.
3. applying hash functions on keys, bucket indexes are inserted into the MinMaxSketch
4. keys 被转换成增量 increments, 用 delta keys 来表示
5. binary encoding

Decode Phase

1. The delta keys are recovered to the original keys
2. recovered keys 被用来 query the MinMaxSketch
3. 从 Sketch 中拿到每个 value 的 bucket index
4. 通过用 bucket index query bucket value 恢复 value

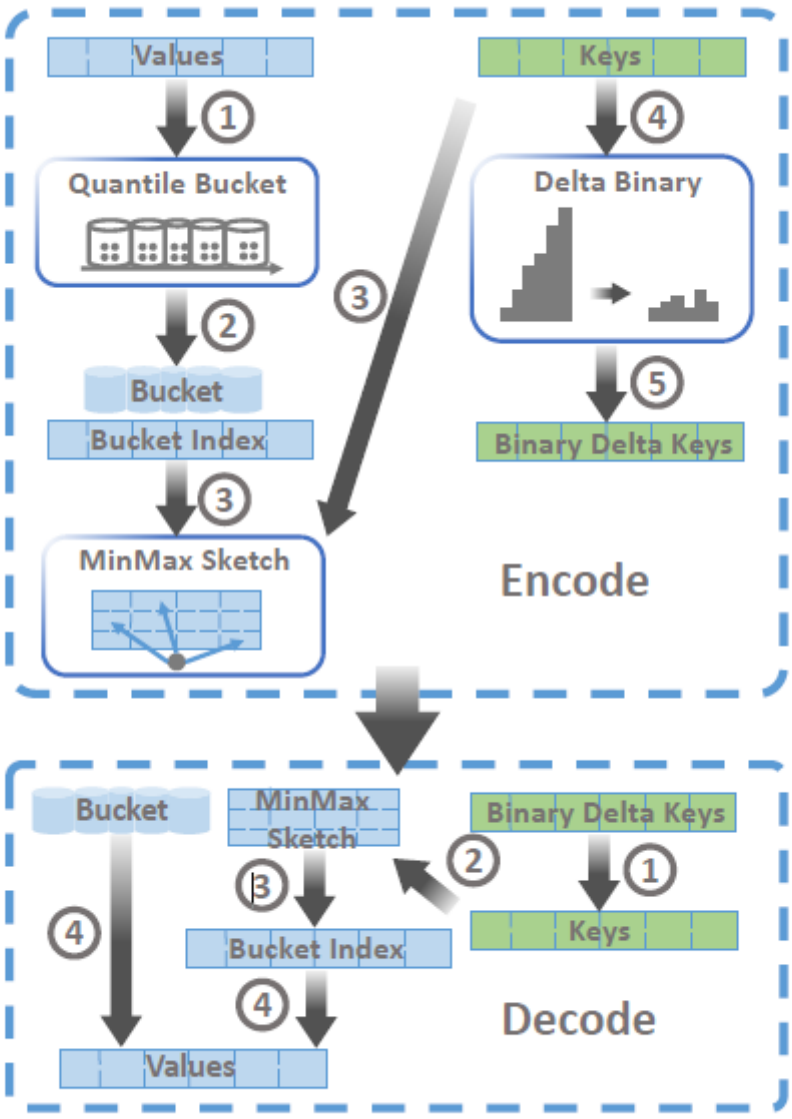


Figure 2: Framework Overview of SketchML

3.2 Quantile-Bucket Quantification

Motivation

不同于 gradient keys 都是 integer, the gradient values 是 floating-point numbers

研究表明, gradient optimization algorithm 是能够在 noise 存在的情况下正常运作

考虑 optimization algorithm 是否能在量化的 low-precision gradients 情况下收敛

与 unpredictable noises 相比，由 quantification 引起的 error 通常是 controllable and bounded

Quantification Choices

当前的 quantification methods 大多数是采用了 uniform strategy, floating-point numbers 被线性地映射为整数 integers.

如图所示， gradient values 通常是服从 a nonuniform distribution 而不是 a uniform distribution.

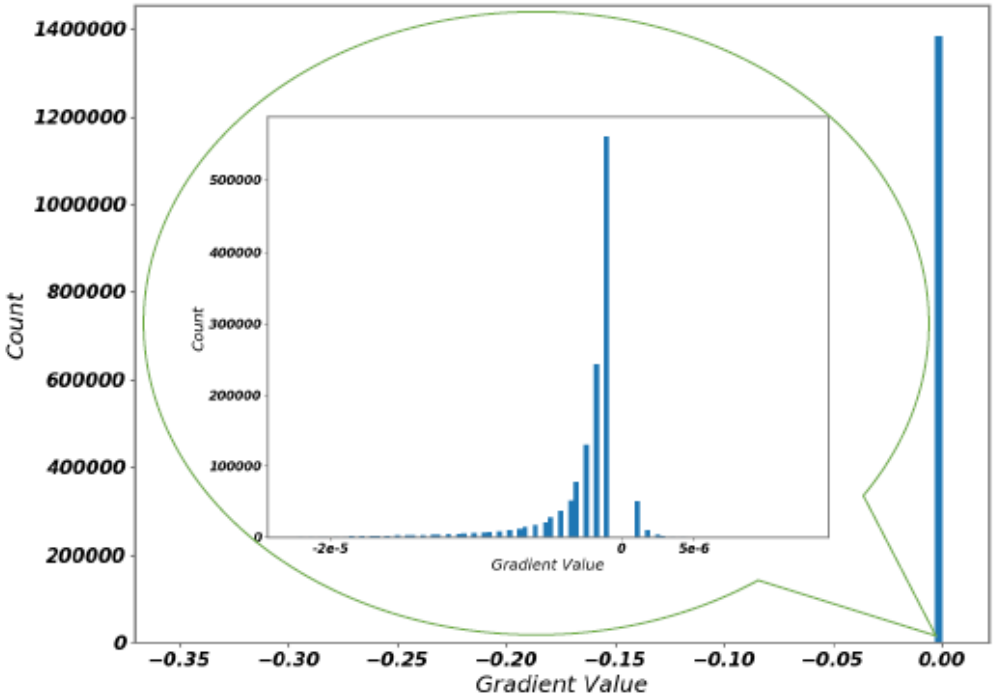


Figure 4: Nonuniform Gradient Values

uniform quantification 是等价地划分 gradient values，并且不能捕获 nonuniform distribution

大多数的 gradient values 是接近0，ZipML直接量化为0，这导致许多的 gradient values 是被忽略的，导致了一种更慢的收敛

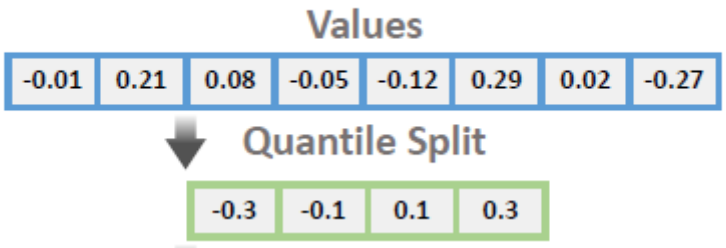
quantile-bucket quantification：

Step 1: Quantile Split

- build a quantile sketch
- generate splits

1. scan all the gradient values, insert into a quantile sketch
2. 从 quantile sketch 中拿到 q quantiles, $\{0, \frac{1}{q}, \frac{2}{q} \ldots, \frac{q-1}{q}\}$
3. 在两个连续的 splits 之间的 quantiles 的个数是 $\{\frac{N}{q}\}$

   划分 items 是通过 number 而不是 value, 每两个 splits 之间的 interval 有相同数量的 gradient values.***(***不能均匀划分怎么办***)***

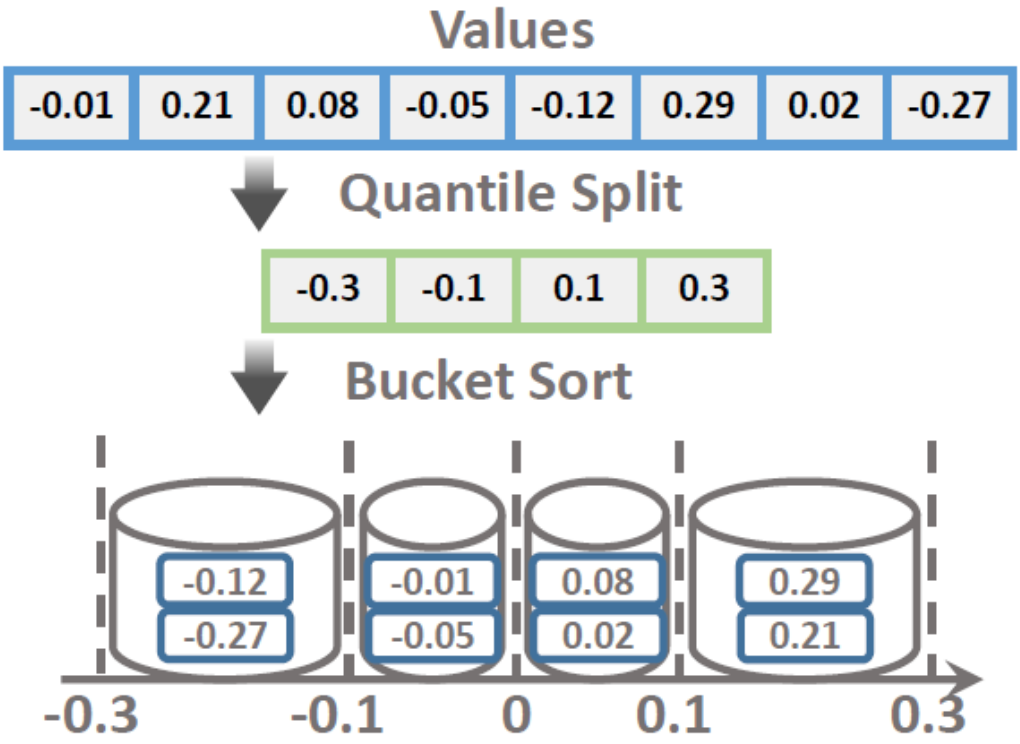


Step 2: Bucket Sort

在得到 quantile splits的情况下，用 bucket sort 去 quantify the gradient values

1. 两个 interval 之间的 splits 被记作 a bucket.

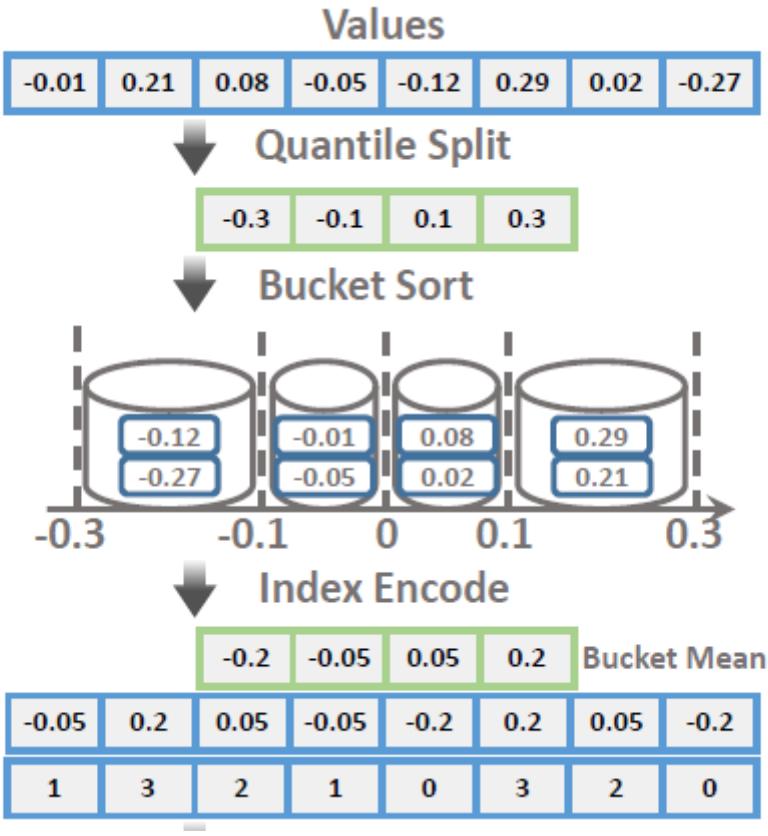
   每个 bucket 都有一个 lower threshold 和 一个 higher threshold.
2. 根据每个 bucket 的 thresholds, 每个 gradient value 都是属于一个特定的 bucket.
3. 每个 bucket 被用 mean value 代替, 两个 splits 的 average
4. 每个 gradient value 被对应的 bucket mean 来代替，这就引入了 quantification errors.



Step 3: Index Encode

在使用 bucket mean values quantify gradient values，但是依旧是存储的 floating-point numbers.

对 bucket 的 mean value encoding，作为 bucket 的 index.



Step 4: Binary Encode

通常情况下, buckets 的数量是一个 small integer.

通过 encoding to binary numbers 来压缩 bucket indexes.

q = 256时候，使用一个字节就可以，这种情况下，空间可以从 8d bytes 压缩到 d bytes

（此处为什么是***8d bytes***）

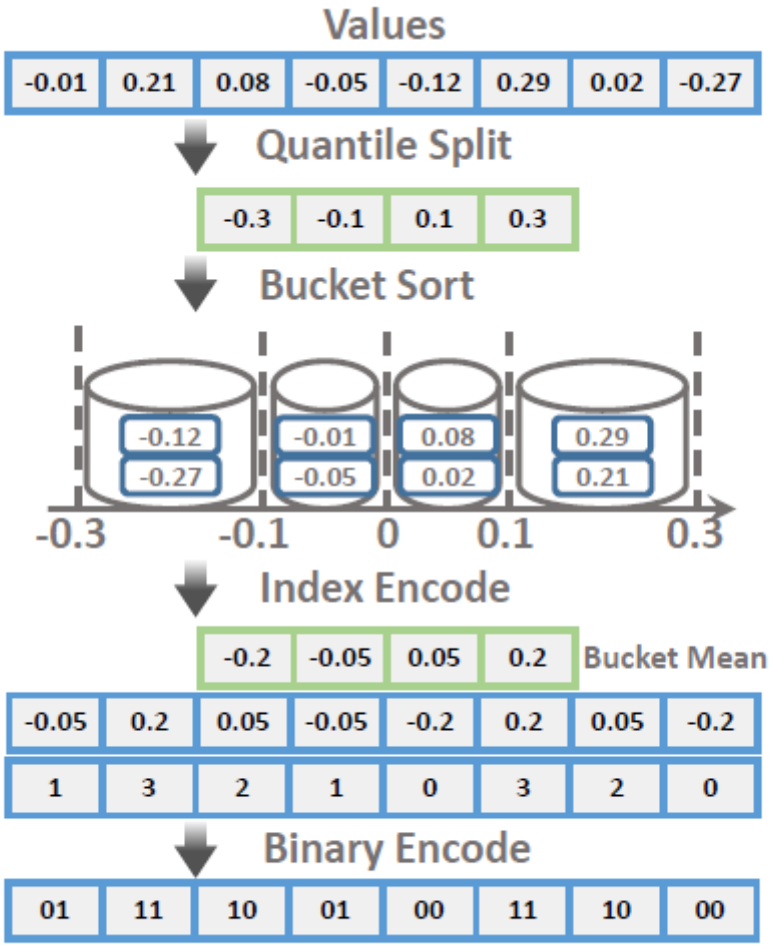


Figure 3: Quantile-Bucket Quantification

3.3 MinMaxSketch

quantile-bucket quantification 有一个 8 倍的 compression rate.

Motivation

Quantile-Bucket Quantification 已经把 gradient values 转换到 bucket indexes.

Frequency sketch 旨在一系列的 items, 每个都是有可能重复出现的

猜测 an item 的 frequency, 而 gradient key 是 no repeated, our goal 是 approximate each single bucket index.

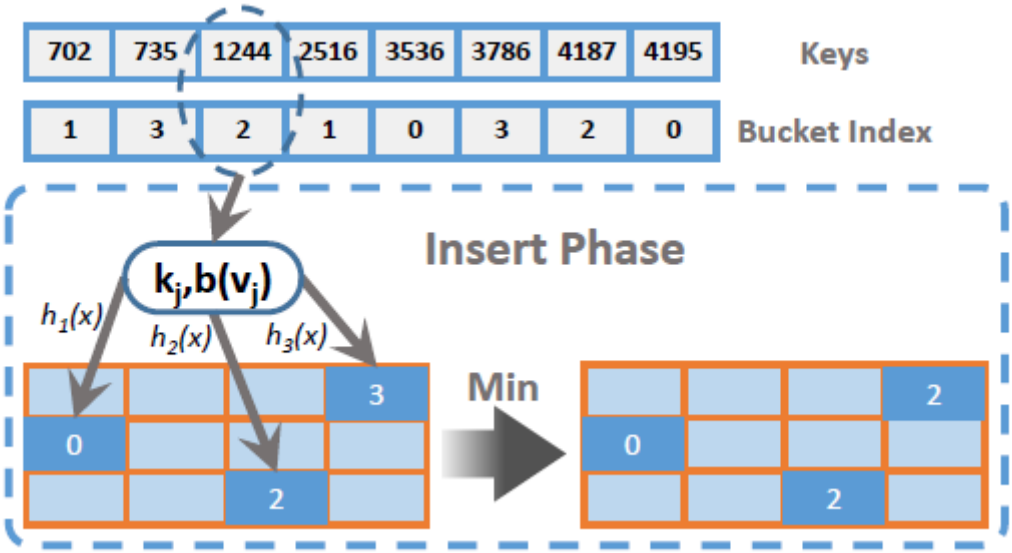
Insert Phase

开始之前, scan all the items and insert them into the sketch.

1. 每个输入项，由 original key 和 encoded bucket index 组成 $(k_j, b(v_j))$
2. 使用 s 个 hash function 去计算 hash codes.
3. 对应于第 i 个 hash table, 比较当前的 value $H(i, h_i(k_j))$ 和 $b(v)j$

如果 $H(i, h_i(k_j) > b_(v_j)$, 则用 $b(v_j)$代替当前值

(在 insert phase 要选择 minimum bucket index)



Query Phase

1. 输入是一个 gradient key k_j，通过 s 个hash function, 从 hash table 中得到了 s 个 hash bin.
2. 从不同的 rows 中得到了 s 个 candidates，选择 maximal one 作为 the final result.

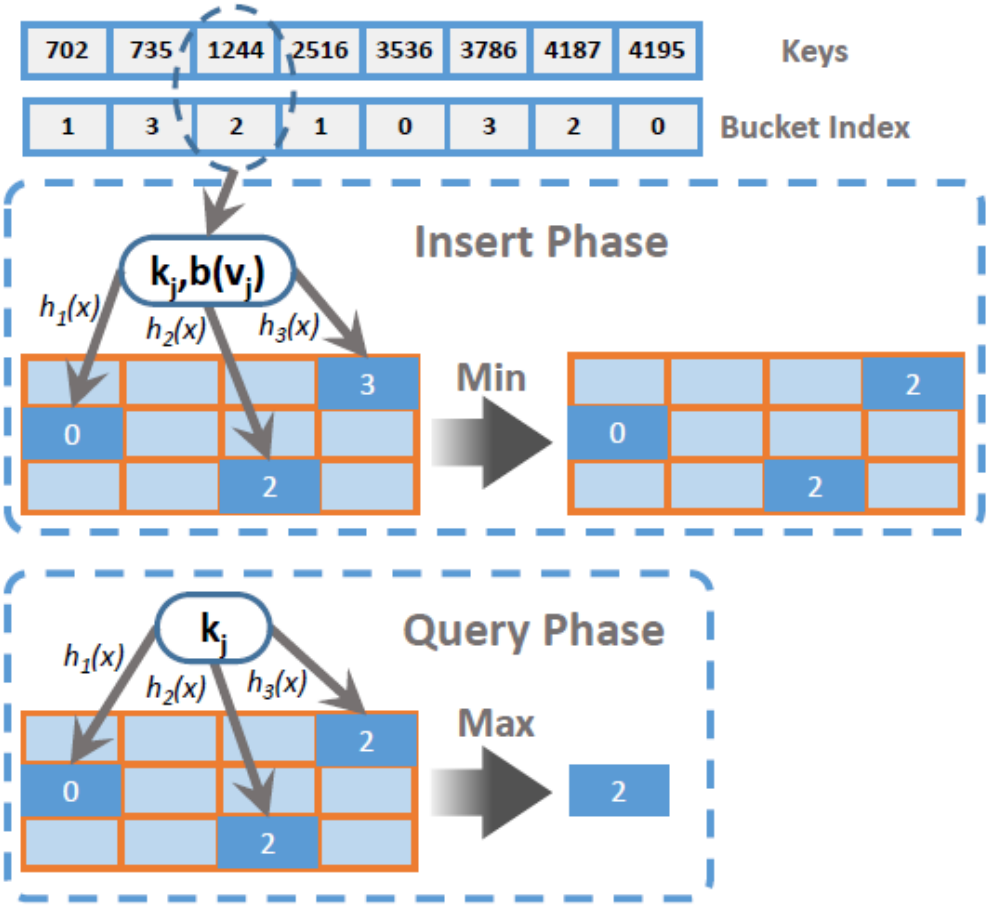


Figure 5: MinMaxSketch

在 insert phase 是选择了 minimum candidate,在 query phase 选择 maximal candidate 可以得到最接近 the original value.

Analysis

problem: the queried result is not guaranteed to be exactly the same as the original value.

在 querying a sketch 时候，有两种 errors

- overestimated error

  引入更大的 query results

- underestimated error

  引入更小的 query results

现有的 frequency sketches 都有 overestimated error, gradient value 也会被放大，这会导致一个不可预测以及不稳定的收敛性

MinMaxSketch 引入的是 underestimated error, 因此 hash bin 不会大于 bucket index, 得到的 gradient value 也是 underestimated.

> 分析：减小 gradient 的规模显然会 slow down the convergence rate, 但是仍然在 correct convergence track.
>
> 而不受控制的增大 gradients 的规模可能会跳过最优点。

Problem 1: Reversed Gradient

MinMaxSketch 会带来 decayed gradient

the bucket index is decayed, the gradient value is uncertain.

因为需要使用 bucket index 去 query bucket mean value, 在这个过程中 the gradient value 的符号可能 be reversed.

- **Case 1**: 0.01 被编码为-0.01，在这种情况下，即使正确 decodes bucket index，它也仍然反转了0.01的符号
- **Case 2:** 对于第5个bucket的0.14，如果MinMaxSketch 生成一个更小的 bucket index, the queried value 变成了 -0.09

梯度优化算法（SGD）对于 decayed gradients 是 robust, 但是对于 reversed gradients 效果不佳

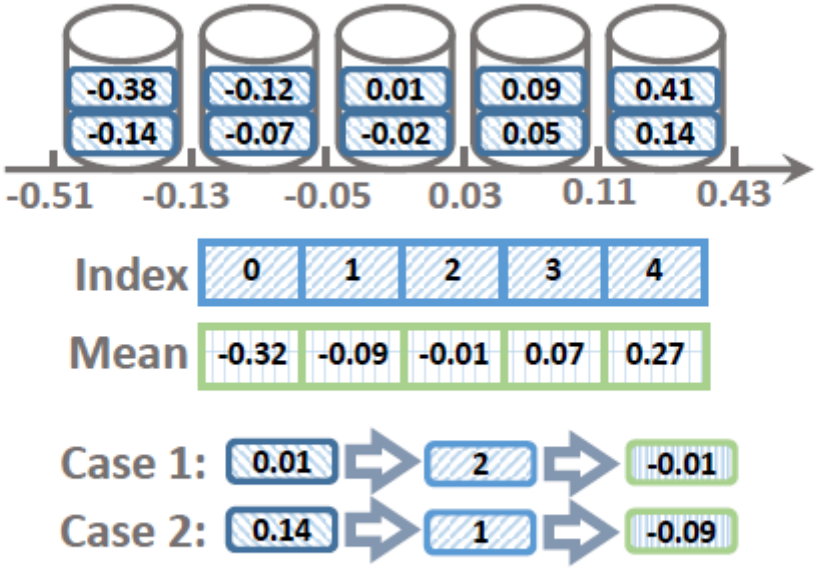


Figure 6: Reversed Gradient

Solution 1: Separation of Positive/Negative Gradients

problem 1 的问题是对 positive and negative gradient values 是一起量化的

因此，需要对于正负梯度分别处理

1. 对于正负梯度，分别使用两个 quantile sketches, 并且使用各自的 buckets 去量化
2. 构建一个 positive MinMaxSketch 和一个 negative MinMaxSketch.
3. insert phase, 为了实现decaying gradient 的效果，选择的是 minimum bucket（the first bucket for positive gradients and the last bucket for negative gradients）

Problem 2: Vanishing Gradient

MinMaxSketch 带来的 decayed gradients, 被称为 vanishing gradient, 尽管 convergence 的正确性不受影响，但是由于每轮step 的减小， convergence rate 也会不可避免的减少

Solution 2: Adaptive Learning Rate and Grouped MinMaxSketch

两种方法去 compensate the problem of vanishing gradient,

Adaptive Learning Rate

Adam algorithm

Grouped MinMaxSketch

原始的 MinMaxSketch 对于 the original bucket index 和 the decoded bucket index 的差距可能会达到 q （quantile splits 的数量）

当 trained model 接近 optimality，the gradient 就会非常小

adaptive learning rate 不能够弥补 bucket index 的变小

因此，将all buckets 划分为 r groups，为每一个创建一个 MinMaxSketch

可以将bucket index 的 最大 error 从 q 降低到 $\frac{q}{r}$

Summary

- MinMaxSketch 是用来压缩在 quantile-bucket quantification 过程中生成的bucket index
- MinMaxSketch 是通过在 insert phase 的 a min protocol 和在 query phase 的 a max protocol 来解决 hash collision.
- 改进：进一步解决了decoded gradients 的 reversal 和 decay 问题

3.4 Delta-Binary Encoding

compress the gradient keys

Motivation

为 gradient keys 设计一个 lossless compression.

gradient 的特征：

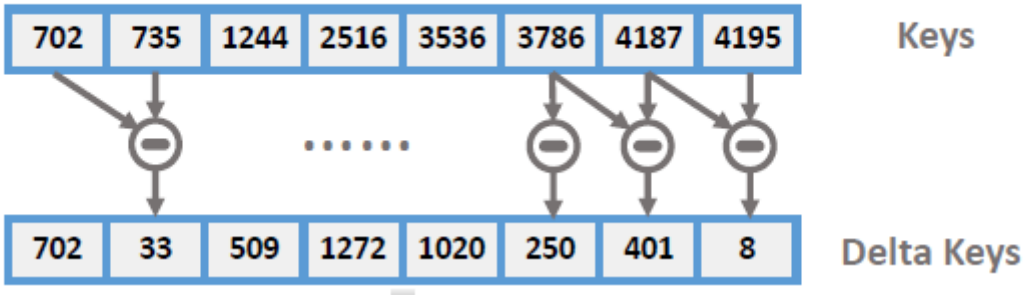
1. keys are not repetitive
2. keys are ordered in an ascending order
3. keys 可能会非常大，但是neighboring keys的 difference 会小很多

基于此，提出仅存储 keys 的 increment.

Step 1: Delta Encoding

the gradient keys 被有序存储在数组中，从后向前扫描并计算两个相邻 keys 的 difference.

得到的是keys 的 increment，即 delta keys.



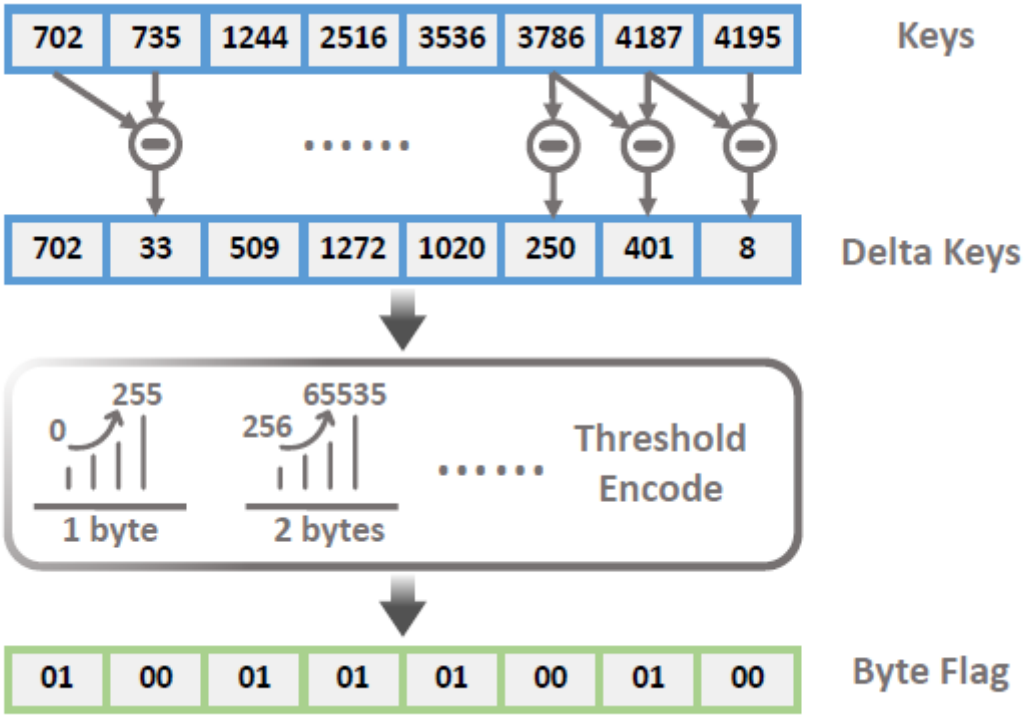
Step 2: Binary Encoding

delta keys 不能用整数或者长整数的格式去存储，这样导致的 memory space and communication cost 是不变的

solution：为不同的 delta keys 分配不同的 spaces，并且用 binary format 编码

每个delta keys 需要的 bytes 被编码为一个二进制数，即 byte flag.

(one byte is 00, two byte is 01)



最后, delta keys 根据 byte flags 被编码为 binary number.

现有的用于压缩整数的方法RLE、Huffman Coding 都是需要有重复性的整数，因此对于 non-repetitive gradient keys 是 useless.

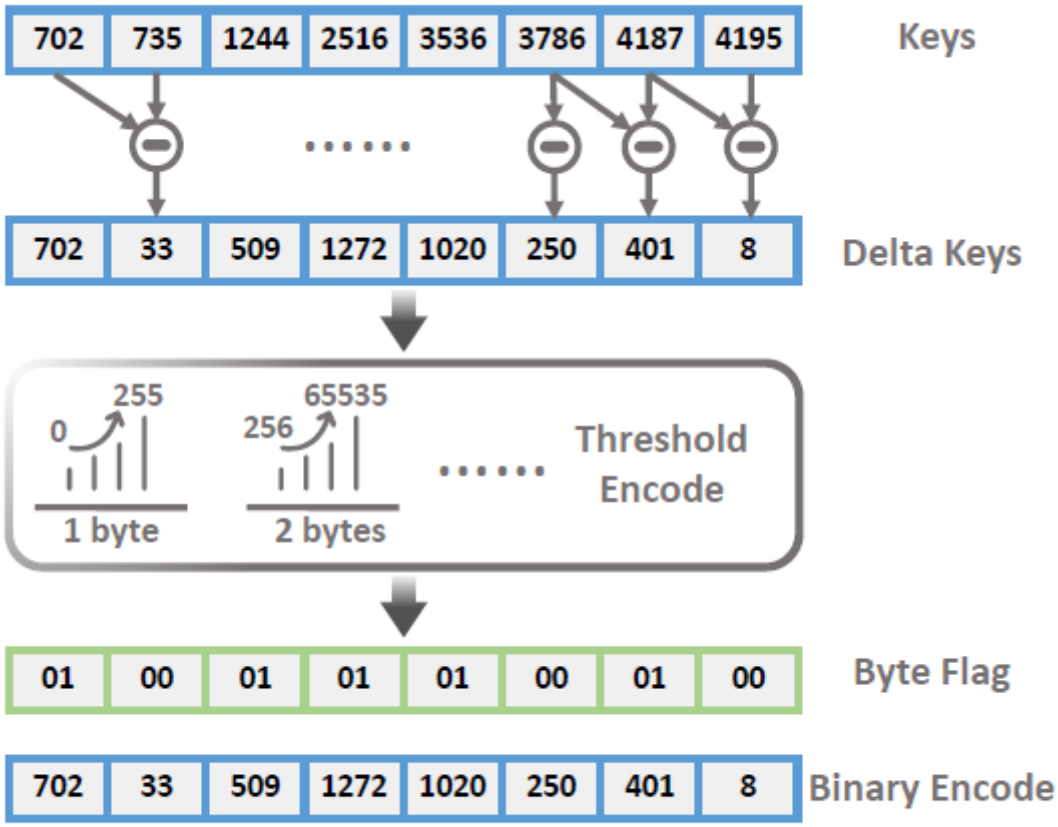


Figure 7: Delta-Binary Encoding

Summary

为了无损压缩 gradient keys，存储的是 keys 的 increment, 并使用 threshold 机制将 increment keys 编码为二进制格式

key-value gradient pair (k_j, v_j) 被转换为 $(\triangle k_j, v_j)$

3.5 Analysis of Space Cost

- **Quantile-Bucket Quantification**

  bucket 需要的 mean values 需要被传送，共 8q bytes (通常，q 是一个小整数)

- **MinMaxSketch**

  r 个 grouped sketches.

  每个individual MinMaxSketch 是 $\frac{s\times t}{r} \times \lceil log_{256} q \rceil = \frac{s\times t}{r} \times \lceil \frac{1}{8} log_2 q \rceil$

- **Delta-Binary Encoding**

  每个 delta key 需要的 bytes 是 $\lceil \frac{1}{8} log_2 \frac{rD}{d} \rceil$

  byte flag 需要 $\frac{1}{4}$ 个byte

4. Experiments

4.1 Experiment Setting

Implementation

executor: 读取 subset, 计算 gradients

driver: aggregates gradients from executors, update the trained model, and broadcasts the updated model to the executors.

Clusters

Cluster-1: ten-node cluster, each machine 32GB RAM, 4 cores, 1-Gbps Ethernet.

Cluster-2: 300-node cluster, each machine 64GB RAM, 24cores, 10-Gbps Ethernet.

Datasets

1. KDD10: 19 million instances and 29 million features.
2. KDD12: 149 million instances and 54 million features.

   预测用户是否会关注社交网站推荐的项目
3. CTR

   预测广告的点击率

Dataset	Size	# Instance	# Features
KDD10	5GB	19M	29M
KDD12	22GB	149M	54M
CTR	100GB	300M	58M

Table 1: Datasets

Statistical Models

三个l_2 regularized models:

- Logistic Regression (LR)
- Support Vector Machine (SVM)
- Linear Regression (Linear)

loss functions are as follows:

$$LR: f(x,|y,\theta) = \sum_{i=1}^{N} log(1 + e^{-y_i \theta^{\mathrm{T}} x_i}) + \frac{\lambda}{2}||\theta||_2$$

$$SVM: f(x,y,\theta) = \sum_{i=1}^{N} max(0, 1 - y_i \theta^{\mathrm{T}} x_i) + \frac{\lambda}{2}||\theta||_2$$

$$Linear: f(x,y,\theta) = \sum_{i=1}^{N} (y_i - \theta^{\mathrm{T}} x_i)^2 + \frac{\lambda}{2}||\theta||_2$$

Adam SGD 进行训练：

β_1 and β_2 是两个接近 1 的 hyper-parameters.

m 和 v 被用来更新训练模型

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} m_t$$

Baselines

- SketchML
- Adam SGD

结合了 momentum and adaptive learning rate (无压缩)

- ZipML

fixed-point quantification method 将 gradient values 压缩为 integers.

Metrics

- the average run time per epoch.
- the loss function with respect to the run time.

Protocol

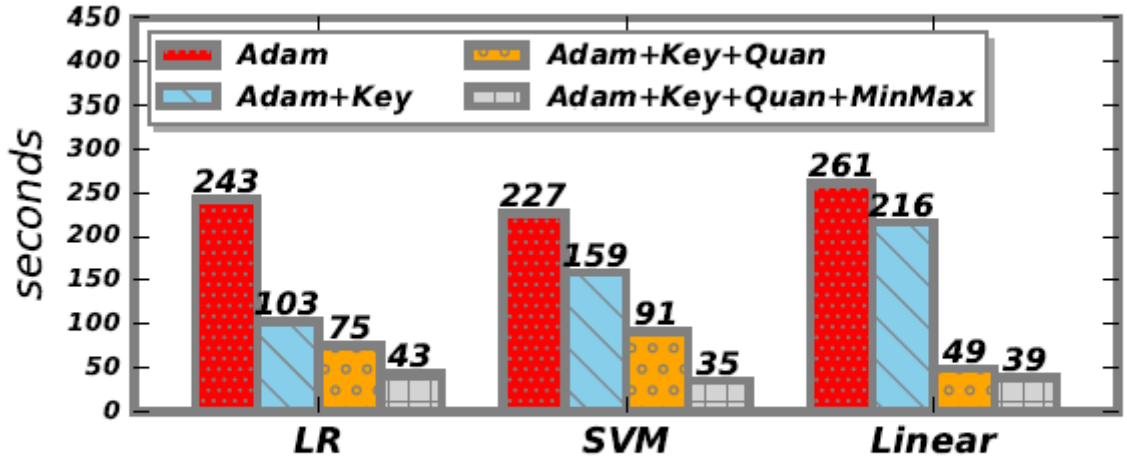
input data 75% 作为 train dataset，25% 作为 test dataset.

采用 SGD. 使用 a batch of instances 来达到 convergence robustness and convergence speed 的 tradeoff.

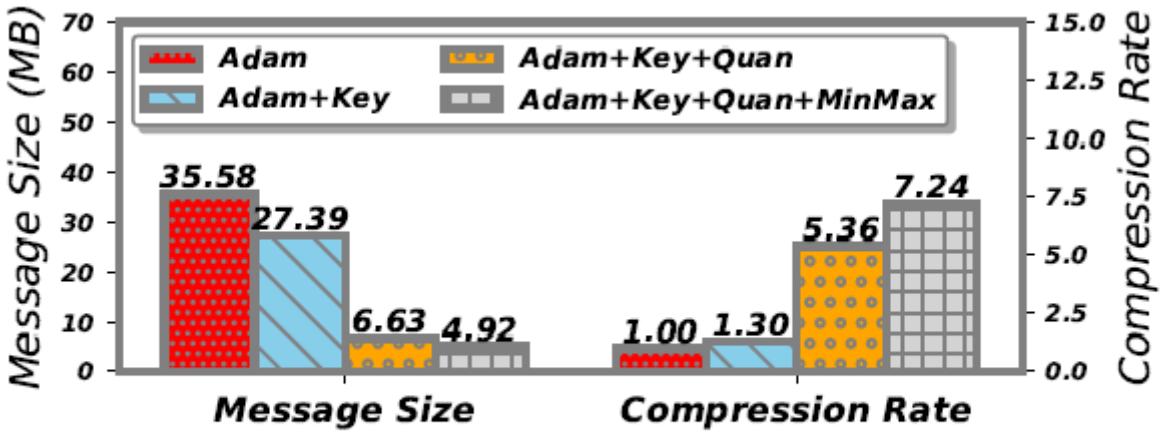
进一步，可以使用 mini-batch SGD 可以降低同步频率并且节省大量的 communication cost.

4.2 Efficiency of Proposed Methods

在 KDD10 dataset 的 10 executors of Cluster-1 执行



(a) Run Time Per Epoch



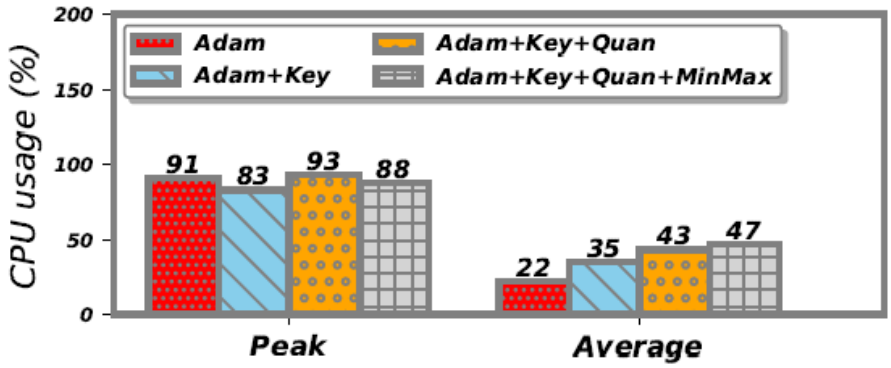
(b) Message Size and Compression Rate

通过改变 batch size ratio 来改变 sparsity.

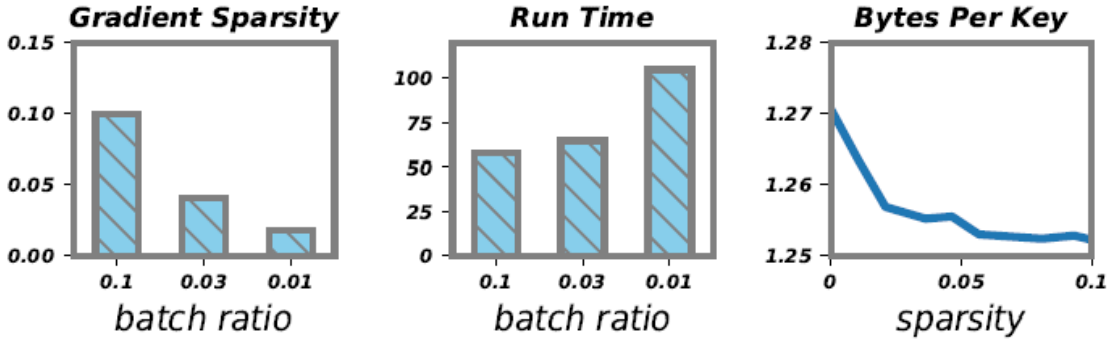
越小的 batch size ，导致了更频繁的通信，增加了每个 epoch 的运行时间。

当 data sparsity 接近0 的时候，需要1.27 bytes。

与原始的 4 bytes 相比， delta-binary encoding 实现了更好的压缩性能。



(c) CPU Overhead



(d) Impact of Batch Size and Sparsity

Figure 8: Effeciency of Proposed Methods. The evaluated metric is the run time per epoch. Adam refers to the basic method without our methods. Key refers to the component of delta-binary encoding. Quan refers to the component of quantile-bucket quantification. MinMax refers to the component of MinMaxSketch.

The results demonstrate that our proposed methods are efficient in reducing the data movement through network.

4.3 End-to-end Performance

在 Cluster-2 上面执行 datasets KDD12 and CTR, 三个算法 （SketchML, Adam, ZipML）, 三个 models (Logistic Regression, SVM, Linear Regression)

将 end-to-end performance 划分到 per epoch 的 run time and the loss of run time.

Logistic Regression

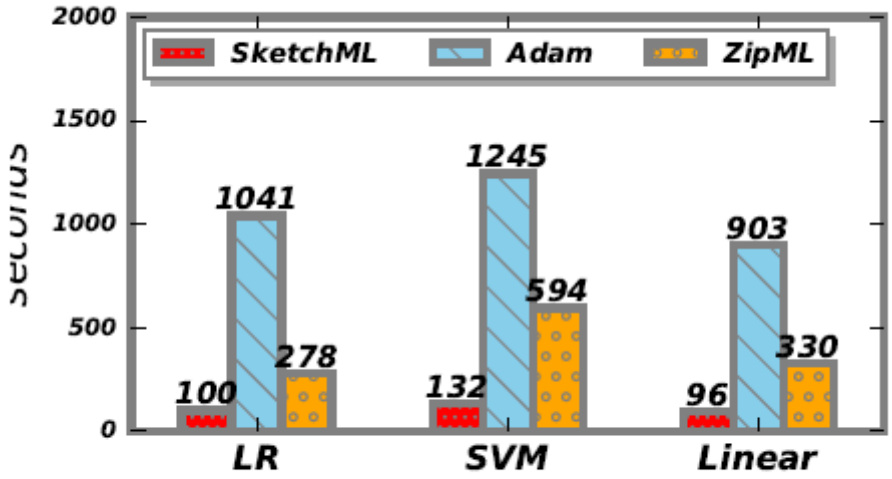
SketchML是比Adam 和 ZipML 快很多
Adam 没有压缩的情况下通信，因此 Adam 是最慢的

ZipML 比 Adam 快 3.7 倍， 是因为压缩了 gradient values.

ZipML 不能压缩 gradient keys.

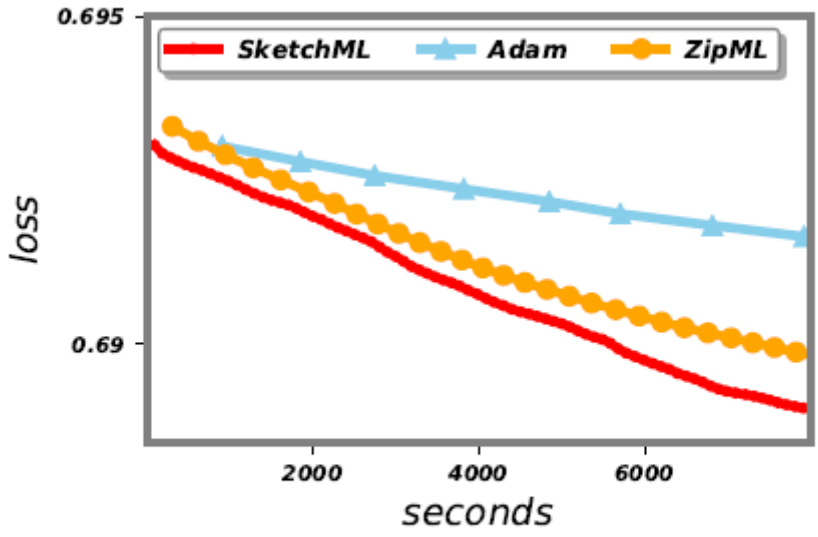
SketchML 比 Aam 快 10.4 倍，比 ZipML 快 2.8 倍，提升来源于减少了在 network 中传输的 gradient data.

而且这种提升会随着 executors 的增多更加显著 （因为通信更多了）



(a) KDD12 Dataset

收敛率方面：SketchML 是最快的，Adam 是最慢的，因为Adam 完成一个epoch需要花费更长的时间，ZipML 比 Adam 也是快很多



(a) Logistic Regression, KDD12

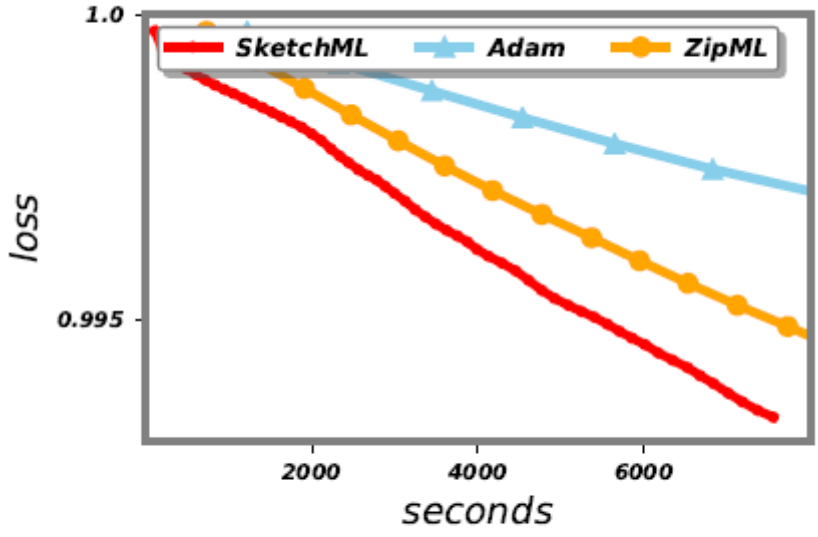
Support Vector Machine

Adam 最慢，ZipML 次之

SketchML 比 Adam 和 ZipML 分别提升了9.4倍和4.5倍

从 (b) 可以看出，随着时间的增长，SketchML 比 ZipML 优势增长更加明显。

因为ZipML 是将许多小的 gradients 量化为 zero，但是随着训练的进行，模型趋于最优，梯度也变得更加小，因此，ZipML 的收敛速度会变慢



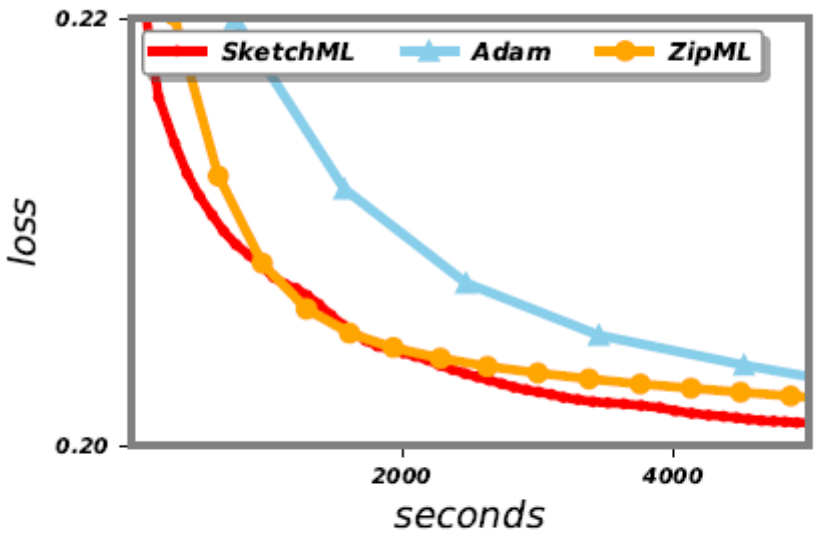
(b) Support Vector Machine, KDD12

Linear Regression

at the beginning of the training process, SketchML 比 ZipML 表现更好

之后，ZipML 在一段时间间隔内短暂优于 SketchML

但是，当模型趋于最优时，由于 ZipML 的量化带来的 error，SketchML 在整体性能上优于 ZipML



(c) Linear Regression, KDD12

4.3.2 Results on CTR Dataset

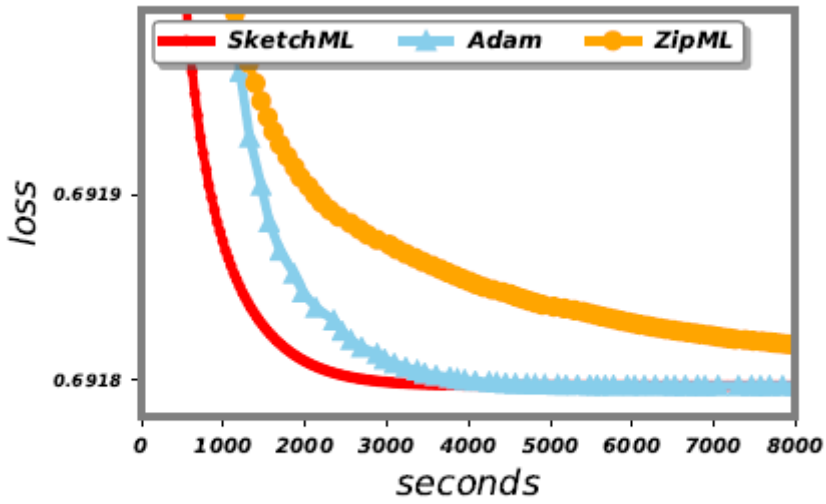
CTR dataset，在 Cluster-2 上，使用 50 executors，给driver 和 executor 分配8GB memory.

Logistic Regression

SketchML 比 Adam 和 ZipML 快了 3.8 倍和 2.7 倍

在CTR 上 SketchML 带来的加速，比在 KDD12 上要小，因为 KDD12 更加的稀疏化

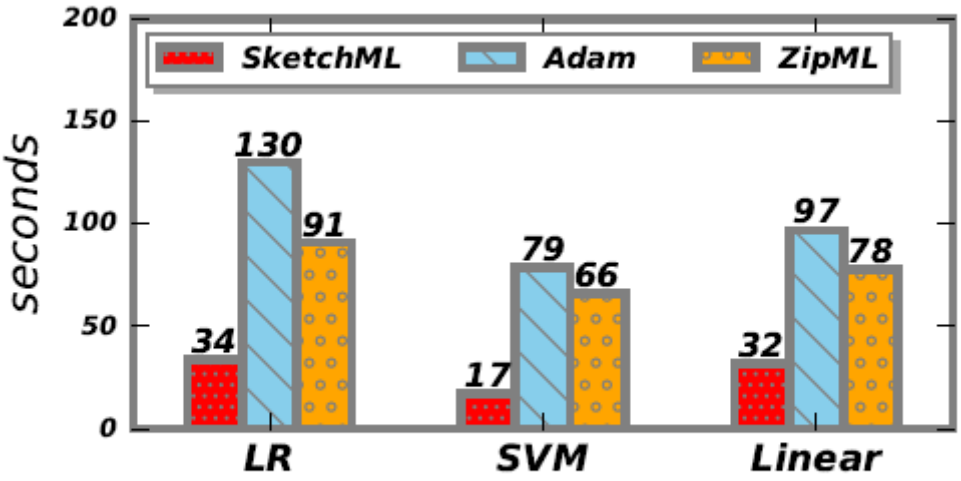
CTR 中，每个 instance 要生成更多的非零梯度，因此带来了更大的计算开销，这对于性能的提升不如在 KDD12 上面对于 communication cost 的提升显著



(d) Logistic Regression, CTR

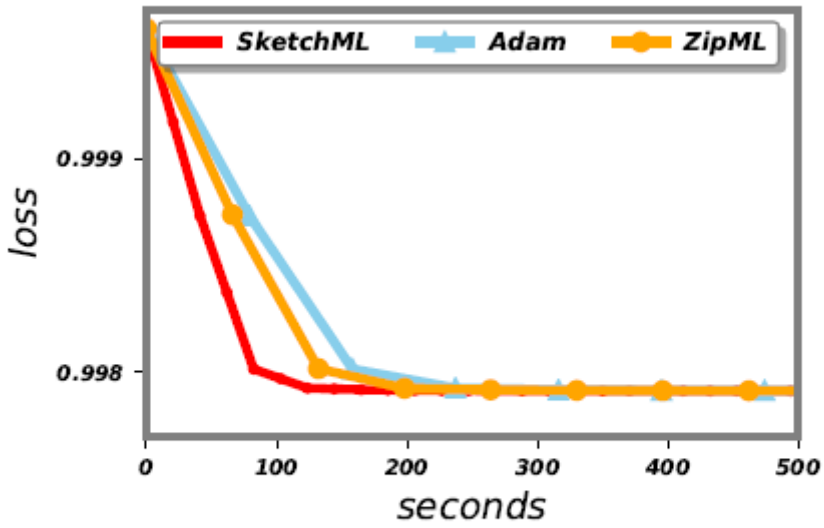
Support Vector Machine

SketchML 比 Adam 和 ZipML 带来了 4.59 倍 和 3.88 倍的性能提升



(b) CTR Dataset

收敛率上，ZipML 由于 faster communication 比 Adam 收敛更快，SketchML 效果最优，能够在较短时间内收敛

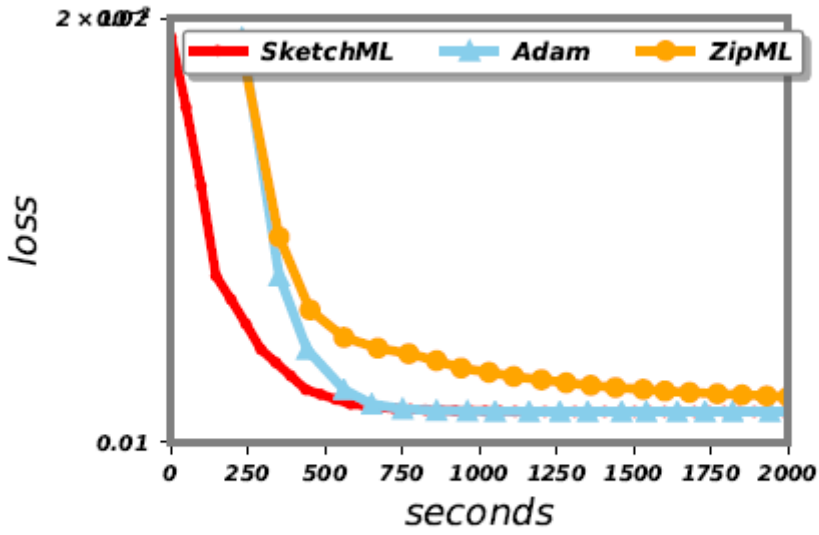


(e) Support Vector Machine, CTR

Linear Regression

收敛率上，ZipML 比 Adam 更慢，这是由于统一量化uniform quantification 的缺陷

SketchML 是最快的，而且能够在短时间内达到收敛



(f) Linear Regression, CTR

4.4 Model Accuracy

在 KDD12 dataset 上测试收敛性

（当loss 在 5 个 epochs 的变化在1%的范围内，则认为是达到了收敛）

三个方法都能达到基本相同的收敛质量，但是SketchML 比另外两种方法要快很多

MinMaxSketch 会导致 underestimated gradients, 这可能会导致收敛速度变慢，但是，dynamic learning rate 和 the grouping strategy 可以解决这个问题

	SketchML	Adam	ZipML
LR	0.6885 / 8.1h	0.6885 / 23h	0.6887 / 11h
SVM	0.9784 / 4.9h	0.9785 / 23h	0.9788 / 10h
Linear	0.2111 / 4.8h	0.2109 / 22h	0.2111 / 9.4h

Table 2: Model Accuracy. The metric is minimal loss against converged time, separated by symbol "/". Run time is in hours.

4.5 Scalability

通过改变 workers(excutors) 的数量来测试 cluster size 对性能的影响

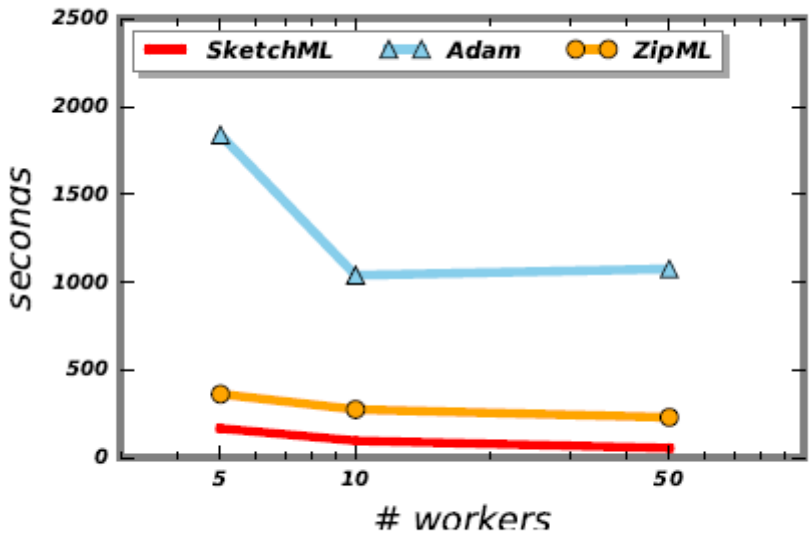
在 KDD12 数据集上测试，评估每个 epoch 的平均运行时间

Logistic Regression

随着workers的增加，三种方法的性能都提升了

但是，50个workers的时候，Adam性能反而退化了，原因是 communication cost 远大于了 computation cost.

对比之下，SketchML 和 ZipML 都有较好的性能提升

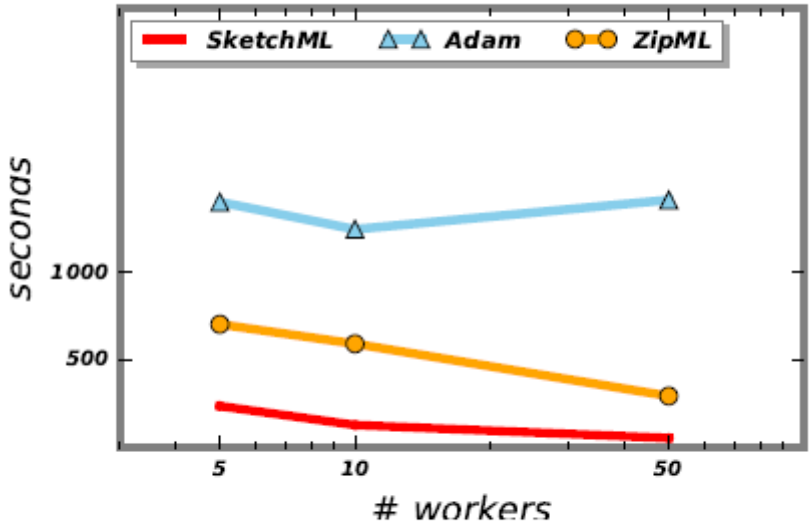


(a) Logistic Regression, KDD12

Support Vector Machine

与 LR 类似，当使用 50 workers 的时候，Adam性能变差

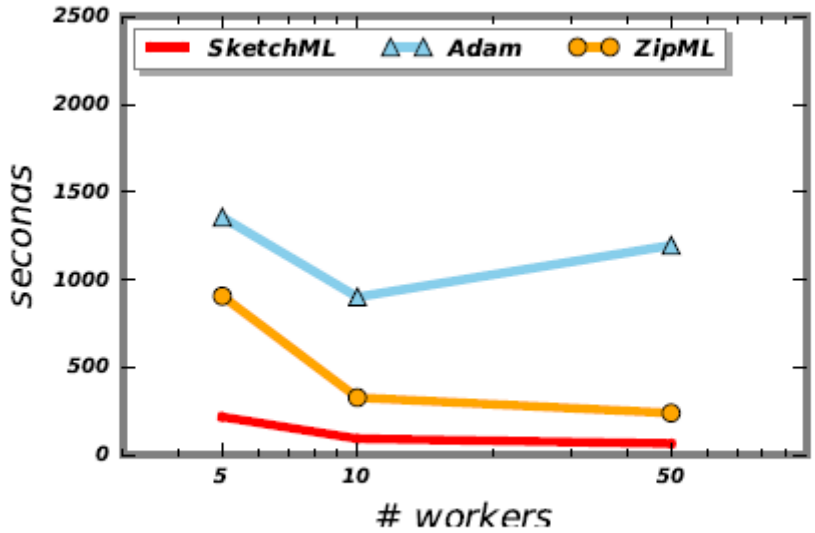
但是，SketchML 和 ZipML 分别可以达到2.3倍和2倍的性能提升



(b) Support Vector Machine, KDD12

Linear Regression

效果同上。



(c) Linear Regression, KDD12

Limitation

1. For dense gradients, the value compression still works,
   but the key compression is redundant
2. For computation-intensive workloads, the benefit of compression is not so significant

5. Related Work

随着data size 和 model size 的增加, aggregating gradients 成了主要的瓶颈

许多研究针对 compress the gradients 来解决 communication cost.

Lossless compression 不能用于 floating-point gradients.

Lossy methods将 floating-point data 转换为 low-precision 的表示

Lossy methods

- **threshold-based truncation**

  encode floating-point data to one bit

  （但是这样过于 aggressive, 丢弃了很多 gradients，收敛性较差）

- **quantification based method**

  eg. ZipML, 根据原始数据的值的范围，将 floating-point number 转换为一个 integer.

  实现了 efficient compression and correct convergence 的 tradeoff, 但是不能满足大规模的 ML cases.

  (1) transferred gradients are sparse and high dimensional.

  为了节省空间，使用 key-value pairs 的方式存储 nonzero elements.

  key--->gradient dimension, value--->对应的 dimension value

  现有的量化方法，只是压缩 gradient values，具有局限性。

  (2) gradient values' distribution is nonuniform

  当前的量化技术都是假设了data 是 uniformly distributed. 许多的 gradient values 被量化为0，导致了 large quantization error.

data sketch algorithm

- **quantile sketch**

不同于quantification, quantile sketch 将 value range 划分为几个 interval，每个 interval 包括相同数量的 items.

- **frequency sketch**

用来估计当前 items 的 frequency.

contribution

使用 sketch algorithms 去压缩 floating-point gradient 到 a low-precision representation

6. Conclusion

1. 使用 quantile sketch and a bucket sort 将 gradient values 用 bucket index encoded 后的更小的binary number 表示
2. MinMaxSketch algorithm 去压缩 bucket indexes.
3. delta-binary method 去 encode gradient keys.
4. 实验表明，SketchML 比目前最快的方法提升了10倍以上。