Huffman Coding Based Encoding Techniques for Fast Distributed Deep Learning

Rishikesh R. Gajjala[1], Shashwat Banchhor[1], Ahmed M. Abdelmoniem[1*]
Aritra Dutta, Marco Canini, Panos Kalnis
KAUST

ABSTRACT

Distributed stochastic algorithms, equipped with gradient compression techniques, such as codebook quantization, are becoming increasingly popular and considered state-of-the-art in training large deep neural network (DNN) models. However, communicating the quantized gradients in a network requires efficient encoding techniques. For this, practitioners generally use Elias encoding-based techniques without considering their computational overhead or data-volume. In this paper, based on Huffman coding, we propose several *lossless encoding techniques* that exploit different characteristics of the quantized gradients during distributed DNN training. Then, we show their effectiveness on 5 different DNN models across three different data-sets, and compare them with classic state-of-the-art Elias-based encoding techniques. Our results show that the proposed Huffman-based encoders (i.e., RLH, SH, and SHS) can reduce the encoded data-volume by up to 5.1×, 4.32×, and 3.8×, respectively, compared to the Elias-based encoders.

KEYWORDS

Distributed training, Gradient compression, Quantization, Huffman coding, Elias and Run-length Encoding.

ACM Reference Format:
Rishikesh R. Gajjala[1], Shashwat Banchhor[1], Ahmed M. Abdelmoniem[1] and Aritra Dutta, Marco Canini, Panos Kalnis. 2020. Huffman Coding Based Encoding Techniques for Fast Distributed Deep Learning. In *1st Workshop on Distributed Machine Learning (DistributedML '20), December 1, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3426745.3431334

1 INTRODUCTION

As the DNN models are becoming complex, one of the fundamental challenges in training them is their increasing size. To efficiently train these models, practitioners generally employ *distributed parallel training* over multiple *computing nodes/workers* [5, 11]. In this work, we focus on the data-parallel paradigm as it is the most widely adopted in practice. In this setting, at each iteration, each worker maintains a local copy of the same DNN model, accesses one of the non-intersecting partitions of the data, and calculates its local gradient. The gradient information are exchanged *synchronously* through the network for aggregation and the aggregated global gradient is sent back to the workers. The workers jointly update the model parameters by using this global gradient. However, the network latency during gradient transmission creates a communication bottleneck and as a result, the training becomes slow. To remedy this, gradient compression techniques, such as quantization [2, 21, 49, 51], sparsification [1, 13, 42, 45], hybrid compressors [4, 43], and low-rank methods [47, 48] are used. In this paper, we focus on the gradient quantization techniques.

We are interested in quantization operators, $Q(\cdot) : \mathbb{R}^d \to \mathbb{R}^d$, that produce a lower-precision quantized vector $Q(x)$ from the original vector x. In general, depending on quantization technique, the quantized gradient, $Q(g_t^i)$, resulting from i^{th} worker is further encoded by using one of the existing encoding techniques. For instance, random dithering [2] and ternary quantization [49] use Elias encoding, Sattler et al. [37] use Golomb encoding [15], and C_{Nat} [21] uses a fixed-length 8-bit encoding. In distributed settings, the quality of the trained model may only be impacted due to compression. Moreover, if the encoding is losses, it can help reduce the communicated volume without further impact on model quality.

To elaborate more, let the *total communication time*, T, to be the sum of the time taken for compression, transmission, and decompression. The main goal of encoding techniques is to encode the compressed gradients in compact form as a result of quantization which can further reduce T. To reduce T, in this work, we focus on the lossless encoding component used in gradient quantization and evaluate their effectiveness across a wide range of compression methods. That is, we decouple quantization from the encoding part and explore possible combinations of quantization techniques and lossless encoding.[1] If the effective transfer speed (including any reductions of transfer speed as a result of compression overhead) S, that accounts for network transmission and computation necessary to compression and encoding, is constant for homogeneous training environments such as, data centers or private clusters, then time, T can be translated primarily to the reduction of the communicated data-volume, V as $T = \frac{V}{S}$. Moreover, existing work on quantization, exploit this assumption and employ arbitrary encoding techniques without carefully considering their complexity and the resulting data-volume. To this end, we try to fill this gap and make the following contributions:

(i) Based on the classic Huffman coding, we propose three encoding techniques—(*a*) run-length Huffman (RLH) encoding, (*b*)

*Corresponding author: Ahmed M. Abdelmoniem (ahmed.sayed@kaust.edu.sa).
[1]Equal Contribution.
Rishikesh and Shashwat were with Indian Institute of Technology, Delhi. Significant part of the work is done while they were interns at KAUST.

DistributedML '20, December 1, 2020, Barcelona, Spain

ACM ISBN 978-1-4503-8182-6/20/12…$15.00
https://doi.org/10.1145/3426745.3431334

[1]As long as the encoding is lossless, the convergence of a distributed optimizer (e.g., SGD and its variants) with a quantization strategy $Q(\cdot)$ remains unaffected.

sample Huffman (SH) encoding, and (*c*) sample Huffman with sparsity (SHS); to encode the quantized gradients generated from codebook quantization. For each of these encoders, we calculate their entropy, average code-length, and computational complexity and compare against the state-of-the-art encoding techniques used in gradient compression—Elias and Run-length encoding (RLE).

(ii) We analyze the performance of our proposed encoders on a wide spectrum of quantization techniques [2, 21, 49], on a variety of DNN models (ResNet-20, VGG-16, ResNet-50, GoogLeNet, LSTM), performing different tasks, across diverse datasets (CIFAR-10, ImageNet, Penn Tree Bank (PTB)) and report our findings. The results show that RLH, SHS and SH can reduce the data-volume by up to 5.1×, 4.32× and 3.8× over the Elias-based encoders, respectively. And, sampling-based techniques (i.e., SH and SHS) achieve faster encoding times of up to 2.5× compared to the Elias-based encoders.

To the best of our knowledge, this is the first work that theoretically and empirically dissects the efficiency of different encoders with respect to the communicated data-volume and complexity. Our proposed encoders can also be used to encode the composition of sparsification and quantization as in Q-sparse local SGD [4]. Since compression is worker-independent and involves no inter-worker communication, encoding methods can apply to compression in asynchronous settings [34].

Notations. We denote i^{th} component of a vector x by $x[i]$. By x_t^i, we denote a vector arising from the t^{th} iteration at the i^{th} node. A string of p consecutive zeros is denoted by 0_p.

2 BACKGROUND

Distributed training. In distributed training, during backward pass, each node i, produces a stochastic gradient g_t^i which is aggregated from n workers, via `all-reduce,all-gather,Broadcast` communication in Horovod [40], or `Push-Pull` parameter server architecture like BytePS [34]. The aggregated gradient $\frac{1}{n}\sum_{i=1}^{n} g_t^i$ is used for updating the model parameters in each iteration.

To reduce communicated data-volume, many proposals use sparsification techniques where a subset of the gradient components is communicated [1, 30, 42]. Although this reduces data volume, in most cases it requires modifications and imposes constraints on the communication techniques (e.g., some sparsifiers only work with `all-gather` rather than the faster `all-reduce` collective). In contrast, quantization reduces the bit-width needed for representing the gradient values [1, 30, 42] and to cultivate its benefits an efficient encoding is required. That is, the workers produce quantized stochastic gradient $Q(g_t^i)$ and an encoding technique $C(\cdot)$ is required to produce $C(Q(g_t^i))$ for efficient communication. The next step, which depends on the APIs of the deep-learning toolkits, such as, `TensorFlow`, or `Pytorch`, is to produce a globally aggregated quantized gradient $\tilde{g}_t = \frac{1}{n}\sum_{i=1}^{n} Q(g_t^i)$, by collecting $C(Q(g_t^i))$ from each node and decoding them to $Q(g_t^i)$. Finally, $\tilde{g}_t$ is encoded again to $C(\tilde{g}_t)$ and sent back to each *worker* node. Each node decodes $C(\tilde{g}_t)$ back to $\tilde{g}_t$ and updates the model parameter locally via the optimizer used in DNN training. This process continues until convergence. We formalize this abstraction (from the perspective of the i^{th} *worker*) with SGD as optimizer in Algorithm 1.

Codebook quantization. Codebook quantization is one of the most popular gradient *compression* techniques. In some cases, it uses only binary bits. E.g., the sign compression in [6]. However, in more sophisticated cases, the gradient components are projected into a vector of fixed length (set by the user), such as random dithering. The user can change the codebook length by varying the quantization states, s, and the inclusion probabilities of each component will vary. For a bit-width b and a scaling factor $\delta > 0$, Yu et al. in [51], quantized $g[i] \in [\varepsilon, \varepsilon+\delta]$ with $\varepsilon \in \text{dom}(\delta, b) := \left\{-2^{b-1}\delta, \ldots, -\delta, 0, \delta, \ldots, (2^{b-1}-1)\delta\right\}$ as:

$$\tilde{g}[i] = \begin{cases} \varepsilon & \text{with probability } p_i = \frac{\varepsilon+\delta-g[i]}{\delta} \\ \varepsilon+\delta & \text{with probability } 1-p_i. \end{cases} \tag{1}$$

Different forms of (1) are adopted in [2, 21, 49], by changing the quantization states s, which is the cardinality of the set $\text{dom}(\delta, b)$. For further details on the state-of-the-art *codebook quantization* techniques, we refer to survey by Xu et al. [50].

Algorithm 1: Abstraction for Distributed quantized SGD—From the perspective of the i^{th} *worker*.

Input : Local data $\mathcal{D}_i$, model parameter x_t, learning rate $\eta_t > 0$;
Output : Trained model $x \in \mathbb{R}^d$.
for $t = 1, 2, \cdots$, **do**
  On Worker: Compute a local stochastic gradient g_t^i at t^{th} iteration;
    Quantize g_t^i to $Q(g_t^i)$;
    Encode $Q(g_t^i)$ to $C(Q(g_t^i))$;
  On Master : Decode $C(Q(g_t^i))$ to $Q(g_t^i)$;
    Do `all-to-all` reduction $\tilde{g}_t = \frac{1}{n}\sum_{i=1}^{n} Q(g_t^i)$;
    Encode $\tilde{g}_t$ to $C(\tilde{g}_t)$ and send back to *workers*;
  On Worker: Decode $C(\tilde{g}_t)$;
    Update model parameters locally via: $x_{t+1} = x_t - \eta_t \tilde{g}_t$;
end

2.1 Encoding technique

For communicating the quantized gradients resulting from different codebook quantization, practitioners use different encodings. In the following we define commonly used technical terms, the entropy and average (expected) code-length that characterize the efficiency of an encoding technique.

Entropy and code-length. For k characters with probability distribution $P = \{p_1, p_2, \cdots, p_k\}$, the *entropy* is: $H(P) = -\sum_{i=1}^{k} p_i \log(p_i)$. The *average code-length* for these k characters with corresponding codeword-lengths $\{l_1, l_2, \cdots, l_k\}$ and probability distribution P is given by $\sum_{i=1}^{k} p_i l_i$. The purpose of an encoding technique is to minimize the average code-length. A uniquely decodable code of a given character set is *optimal* if there exists no other uniquely decodable code with a smaller average code-length.

Run-length (RL) and Elias (ELI) encoding. Run-length encoding is a classic *lossless encoding technique* in which data values (or characters) are encoded as tuples containing the particular data-value and its frequency of consecutive occurrences. Elias is an universal coding technique with predetermined codewords. `Elias`(0) and `Elias`(1) are defined as 1 and 0, else, for any $x > 0$, the binary code of `Elias`(x) is given by 0 prepended by binary representation of x prepended by `Elias`(number of bits taken by x in binary code−1).

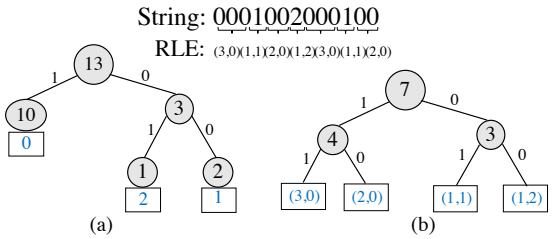


Figure 1: (a) Huffman encoding (16 bits), (b) RLH (14 bits).

2.2 Huffman coding

Huffman coding [22] is a *lossless encoding* which gives prefix codes with optimal average decode-length and uses a greedy algorithm to construct these Huffman codes and store them as a tree. See Figure 1 (a) for an example of Huffman coding of string "0001002000100". We refer to [9] for a detailed algorithm. The following Lemma characterizes the properties of Huffman coding.

Lemma 1. [10] We have, $H(P) \leq CL(P) \leq H(P) + 1$, where $H(P)$ is the entropy and and $CL(P)$ is the average code-length of Huffman codes for a frequency distribution P on characters.

Decoding. To decode the compressed bit stream, we need to have the Huffman tree which can either be constructed or be sent. We traverse the tree in a bit by bit fashion to decode. The decoding speed can be improved by traversing multiple bits called *blocks* at a time [32], by using look up tables [18], and by efficiently using the memory hierarchies [3].

Runtime. Once we have k characters and their frequencies, the run-time for Huffman coding is $O(k \log(k))$. This can be done in $O(k)$ time, if frequencies are given in a sorted order. However, in practice, we need to obtain the frequencies for which we need to take a pass on the entire data.

We realized that codebook quantization comes with an interesting attribute—the quantized components reside in different quantization states. If one considers these quantization states as the characters, then number of items in each state represents their frequencies rendering an ideal scenario to use Huffman coding for communicating those characters. But gradient dimension poses a major challenge even for smaller quantization states as one needs to calculate the frequencies. Therefore, it is unrealistic to build a Huffman tree by following the classical Huffman coding.

Huffman codes for compression. Schmidhuber et al. [39] used Huffman coding to compress text files from a neural predictive network. Han et al. in [17] used a three-stage compression method consisting of pruning, quantization, and finally the Huffman coding to encode the quantized weights (also see [8]). Ge et al. [16], proposed a hybrid model compression algorithm with Huffman coding to encode the sparse structure of the pruned weights. Huffman codes are optimal among all variable length prefix codings. However, Elias [14] and Golomb encoding [15] can take advantage of some interesting properties, such as repeated appearance of certain sequences and achieve a better average code-lengths. To the best of our knowledge, in gradient compression, only SKCompress [23] uses Huffman coding to encode sparse gradient indices. One of the reasons for slow adoption of Huffman encoding is the cost of building the Huffman tree which for modern DNNs, can pose humongous computation overhead. Therefore, one may ask: **Can we achieve a better average code-length based on the classic Huffman encoding and by exploiting certain properties of the quantized gradients?**

3 HUFFMAN BASED STRATEGIES

To answer the above question, we propose using novel encoding techniques based on the classic Huffman coding.

3.1 Run-length Huffman (RLH) encoding

RLH encoding is a hybrid of two classic encoding techniques—RLE and Huffman coding, in which the Huffman coding is used on the string obtained by RLE. E.g., to decode the string "0001002000100" we use RLE to get the following run-length document: (3, 0), (1, 1), (2, 0), (1, 2), (3, 0), (1, 1), (2, 0), and then by using each tuple as a single character, we apply the Huffman algorithm on the character set with corresponding frequencies and obtain the final encodings (see Figure 1 (b)). This encoding technique is also known as modified huffman coding which is standardized by TIFF and commonly used in fax compression [25]. To the best of our knowledge, we are the first to propose a hybrid encoder for encoding compressed gradients in distributed DNN training.

Entropy and code-length. In RLH, the distribution of the characters are obtained by RLE. Therefore, the new distribution of characters after RLE, P_{RLH}, is a function of the original distribution, P. As a result, the entropy of RLH: $H(P_{\text{RLH}}) \leq H(P)$ [10]. This, together with Lemma 1 give: $CL(P_{\text{RLH}}) \leq H(P_{\text{RLH}}) + 1 \leq H(P) + 1 \leq CL(P) + 1$. In practice, the code-length is much lower than this bound. The average code-length of RLH is better than that of Elias. Golomb encoding is also a special case of RLH encoding for only two characters following Assumption 1.

Theoretically, all the encoding techniques take at least $O(d)$ time for $Q(g) \in \mathbb{R}^d$. However, RLH requires another iteration over $Q(g)$ before assigning the codes. In the following, we propose a sampling technique to mitigate the additional overhead of RLH but with a slight increase in the average code-length as shown in tables 3 and 4. Similar to RLH, by using a run-length compression (RLC), Chen et al. in [7] obtained a $1.2\times - 1.9\times$ compression on the convolution layers of AlexNet [27].

3.2 Sample Huffman (SH) encoding

As the gradient dimension d is large, iterating through all the quantized components and calculating their frequencies is not cost-effective. To remedy this, we propose a uniform sampling technique. Sampling techniques are quite popular in gradient compression—E.g. to perform an efficient Top-k selection, [30] used a sampling technique where $r\%$ of the gradient components are chosen first, and then a Top-k selection is made on that sampled set. We sample a set of indices, S from $[d]$, where each S_i is sampled uniformly from a discrete distribution $[1, d]$, such that $S_i \sim \mathcal{U}[1, d]$. We pick the gradient components corresponding to those S_i indices, $i \in [|S|]$, from the quantized gradient vector. Frequencies of the quantized gradient components will be their frequency in those S samples. We construct the Huffman codes using these frequencies.

For implementation, we add 1 to the frequency of all the quantization states, to prevent any of them from being zero. We call this as sample Huffman (SH) encoding.

Entropy and code-length. Let the probability distribution of original qunatized gradient be $P = \{p_1, p_2, \cdots, p_s\}$ and within the sample S of size $|S|$ be $P' = \{p'_1, p'_2, \cdots, p'_s\}$. Then the entropy and average code-length of SH are: $H(P') = -\sum_{i=1}^{s} p'_i \log(p'_i)$ and $CL(P') = -\sum_{i=1}^{s} p'_i l_i$, respectively, where $\{l_1, l_2, \cdots, l_k\}$ are the codeword lengths of the characters with distribution P'. As we use Huffman coding on P', by Lemma 1 we have, $CL(P') \leq H(P') + 1$. As, $|S| \to d$ and $p'_i \to p_i$, we have $H(P') \to H(P)$, and therefore, $CL(P') \to CL(P)$, the average code-length of Huffman.

However, for sparse vectors, Huffman coding is not effective compared to RLE, as it encodes each character. We need to use the sparsity to design an efficient encoding technique.

3.3 Sample Huffman with Sparsity (SHS)

We define the sparsity fraction of a vector p, as the fraction of its components that are 0. Our next assumption is on the independence of the quantized gradient components.

Assumption 1. Quantized gradient components are mutually independent.

The above is a mild assumption. That is, if we know a particular quantized gradient component to be 0, then the probability that the next quantized gradient component to be 0 remains p.

With this assumption, we propose a new encoding technique to give efficient codes for sparse vectors. Let $Q(g)$ be a quantized gradient with sparsity fraction p and s distinct quantization states. The sparsity fraction is estimated by using the sampling technique described in Section 3.2. The probability to encounter n' zeros followed by a non-zero quantization state is $p^{n'}(1-p)$, by Assumption 1. For a given length cap γ, we treat each of these sequences with n' zeros as character $A_{n'}$ for $1 \leq n' < \gamma$. All the sequences of zeros with string lengths greater than or equal to γ have probability of occurrence p^{γ} and can be represented by using multiple A_is. By using these $(s + \gamma - 1)$ characters and their frequencies, we now construct the Huffman tree and define the encoding as sample Huffman with sparsity (SHS). Note that, length cap was also used for run length limited encoding in [36]. Note that our work is valid for any number of quantized states giving the optimal compression and the compression strategy in [37], is a special case of our SHS.

An example. Consider 4 quantized states $\{0, 1, 2, 3\}$ with frequencies $\{0.7, 0.05, 0.05, 0.2\}$ and $\gamma = 5$. The probability of the string 0_5 is 0.7^5 and the rest of the zero strings with length i $(1 \leq i \leq 4)$ have probability $0.3(0.7)^i$. The characters $\{0, 1, 2, 3, 00, 000, 0000, 00000\}$ are given probability $\{0.21, 0.05, 0.05, 0.2, 0.147, 0.1029, 0.072, 0.168\}$ and the encodings will be $\{01, 11110, 11111, 00, 101, 100, 1110, 110\}$.

Entropy and optimal code. For a set of $s + 1$ characters with probability distribution $P = \{p, p_1, p_2, \cdots, p_s\}$, where p is the probability of the occurrence of the character (0), the probability distribution of $s + \gamma$ characters of SHS with length cap γ is $P_\gamma := \{p_1, p_2, \cdots, p_s\} \bigcup \{p^i(1-p) | i \in [1, \gamma - 1]\} \bigcup \{p^\gamma\}$. Therefore, the entropy is $H(P_\gamma)$

$= -p^\gamma \log(p^\gamma) - \sum_{i=1}^{s} p_i \log(p_i) - \sum_{i=1}^{\gamma-1} (1-p)p^i \log(p^i(1-p))$

Table 1: Sample of the encoding time of the quantized gradient

Quantizer	Model	Encoding time (s)					
		Huffman	Elias	RLE	SH	SHS	RLH
RD	ResNet-20	0.20	0.26	0.26	0.11	0.13	0.85
	VGG-16	13.95	18.95	18.83	6.60	7.07	26.02
	GoogLeNet	6.84	9.03	9.68	4.10	4.67	10.83
	ResNet-50	20.80	18.46	18.24	9.04	10.40	32.26
	PTB-LSTM	85.55	53.64	80.68	22.17	32.4	151.94

As $\gamma \to \infty$, we have

$H(P_\infty) = -\sum_{i=1}^{s} p_i \log(p_i) - p \log(1-p) - p\frac{\log(p)}{1-p}$.

The optimal codes for SHS are achieved when $\gamma \to \infty$. By using the above definitions we get the following lemma.

Lemma 2. For SHS with length cap $\gamma > 0$, we have $H(P_\gamma) = H(P_\infty) + (p^{\gamma+1} \log p)/(1-p) + p^\gamma \log(1-p)$.

Our next lemma guarantees that the average code-length of SHS obtained by introducing a length cap is within $\Delta + 1$ bits of $CL(P_\infty)$ and its proof follows from Lemma 1 and 2.

Lemma 3. For SHS with length cap $\gamma > 0$, we have $CL(P_\gamma) \leq H(P_\gamma) + 1 = H(P_\infty) + \Delta + 1 \leq CL(P_\infty) + \Delta + 1$, where $CL(P_\gamma)$ and $H(P_\gamma)$ denote the average Huffman code-length and the entropy of the character set formed with length cap γ respectively and $\Delta = p^\gamma \log(1-p) + (p^{\gamma+1} \log p))/(1-p)$.

3.4 Time complexity Analysis

Suppose $Q(g) \in \mathbb{R}^d$ and has s quantization states. Although it takes $O(d)$ time for all techniques for a single pass over the gradient, the time to assign these code-words is different for each of them. For Huffman coding assigning code-words take $O(s \log(s))$ time and hence the total time will be $O(d + s \log(s))$. Similarly, when there are k distinct characters formed after RLE, RLH will take $O(d + k \log(k))$ time. Note that, k varies between s and $\sqrt{d}$. In SH without sparsity, we sample d' gradient components out of d gradient components, and then determine the frequencies of the s quantization states by using sampled d' components. As a result, it takes $O(d + s \log(s))$ time. For SHS, with a given length cap γ, the time complexity is $O(d + (s + \gamma)\dot{\log}(s + \gamma))$. In practice, $d \gg s, k, \gamma$, so the time complexities remain $O(d)$.

We present a sample of the encoding time values of various encoders applied on the quantized gradient vectors using *RD* quantizer in table 1.[2] The results show that the sampling-based approaches result in the fastest encoding times and RLH is the slowest encoder. However, as shown later, RLH yields the best compression ratio and the proposed Huffman encoders, in many cases, are the second best. Therefore, a trade-off exists between the proposed Huffman encoders that compresses more versus ones with lower time complexity. Specifically, the speeds of SH and SHS encoders are comparable but they can be up to 7× faster than RLH. In contrast, RLH, which is significantly slower, can achieve higher compression ratios of up to 3× on average compared to SH and SHS.

Although all the encoding techniques take $O(d)$ time, Elias encoding need only one pass over the data and classic and run-length Huffman need two passes over the data, and that is $2d$ time. In case of sampling, let $|S| = d' \ll d$. If d' gradient components are selected, the total time taken is $d + d'$ (which is less than Huffman's

[2]Encoding times may vary on different systems. The results are based on a non-optimized implementation ran on a personal computer equipped with low-end CPU.

$2d$ time) and this behaves as good as the Huffman coding. SHS also performs as good as RLH in $d + d'$ time, when the sparsity assumption holds. All the proposed encoding techniques are *lossless*, as they apply lossless Huffman coding on different character sets [22].

Parallel implementation. The encoders in this study are parallelizable [28, 38]. Given n workers, we can partition the gradient into n subvectors for RLH and allow each worker to find the frequencies of its corresponding subvector. We handle the sequences which are partitioned into two different sub vectors by decreasing the frequency of length of each partitioned sequences and increasing the frequency of their combined length, both by one. We can perform a merger at the end of each part, in case a run length sequence is partitioned. Let l_i^e be the length of the run length sequence of the character c_i^e at the end of the i^{th} partition for all $i \in [0, n-1]$. Further let l_{i+1}^b be the length of the run length sequence of the character c_{i+1}^b in the beginning of the $(i+1)^{th}$ partition for all $i \in [1, n]$. Then, if and only if $c_i^e = c_{i+1}^b$, we decrease the frequencies of l_i^e, l_{i+1}^b by 1 and increase the frequency of $l_i^e + l_{i+1}^b$ sequence by 1 for the character c_i^e. The run time to find the frequencies will be $O(d/n)$. For SH and SHS, sampling can be done in n partitions taking (d'/n) time. Huffman tree for s characters can also be constructed in $\log(s)$ time in parallel [28]. With this, the tree construction takes $O(\log k)$ for RLH, $O(\log s)$ for SH, and $O(\log(s+\gamma))$ for SHS. The parallel version of these algorithms can leverage highly-parallel accelerators such as GPU and FPGAs [29, 33, 44].

4 EVALUATION

In this section, we evaluate various encoding schemes covered by our study. The results show that RLH excels over all other encoding techniques in most of the benchmarks. In particular, RLH, SHS and SH can improve the compressed volume by up to 5.1×, 4.32× and 3.8× compared to the commonly used Elias encoding, respectively.

Compression techniques. In this study, we use the following compression techniques: random dithering (RD) [2], ternary quantization (TQ) [49], natural compression (C_{Nat}) [21].

Benchmarks. We implement distributed training using PyTorch [35] as the ML framework and Horovod [40] as the communication library. The compression techniques are implemented using high-level PyTorch APIs. The benchmarks, models, and datasets used in this work are listed in Table 2.

Hyperparameters. We set the local mini-batch size to 160 images per batch for CNN benchmarks on CIFAR-10 and ImageNet datasets and 20 tokens per batch for RNN benchmark on PTB dataset. We set the learning rate to 0.5 for CIFAR-10, 0.05 for ImageNet, and 22.0 for PTB benchmarks. The learning rate is warmed up and grows exponentially during the first 5 epochs until it reaches the chosen learning rate. We run CIFAR-10 experiments for 70 epochs and PTB experiments for 30 epochs. Due to the length of ImageNet experiments, we only run them for 15 epochs. Finally, we use a sample size (d') of 10^4 in all our experiments.[3]. TQ is the only method that requires careful consideration. First, it does not converge without an Error-Feedback (see [42, 43]). Secondly, the learning rate for TQ has to be carefully tuned for each benchmark. We perform trial-and-error tuning for the learning rate that allows for some-level of convergence and so learning rate for TQ is 0.005 for CIFAR-10, 0.01 for ImageNet and 22 for PTB datasets.

[3]We also experiment with smaller sample sizes of 10^2 and 10^3 and we find that both the code-lengths of SH and SHS are larger by up to 1% of Huffman code-length. We leave detailed sensitivity analysis of the sample size to the future work.

Table 2: Summary of the benchmarks used

Task	Neural Architecture	Model	Dataset	No. of Parameters
Image Classification	CNN	ResNet-20 [24]	CIFAR-10 [26]	269,722
		VGG-16 [41]	CIFAR-10 [26]	14,728,266
		GoogLeNet [46]	ImageNet [12]	6,610,344
		ResNet-50 [24]	ImageNet [12]	25,557,032
Language Modelling	RNN	LSTM [20]	Penn Tree Bank [31]	66,034,000

Table 3: Test Perplex. and comp. ratios for PTB benchmark

Baseline Perplexity (↓)	Quantizer	Quantized Perplexity	Compression Ratio (×)					
			Huffman	Elias	RLE	SH	SHS	RLH
103.398	RD	125.955	31	348	80	31	203	491
	TQ	970.335	31	10814	3746	31	855	16432
	C_{Nat}	103.2	24	6	3	24	27	32

Evaluation metrics. We evaluate all encoding techniques by using two metrics—(*i*) the quality of the trained model and (*ii*) the communicated volume to achieve such quality. We report the actual code-length in bits of the communicated gradient (along with the decoding information such as the code-book) in each iteration for different quantization techniques. We also report the training accuracy and the compression ratio which is the code-length of the non-compressed gradient normalized by the code-length (the encoded compressed gradient) . Considering 32-bit as the default floating-point precision in modern DNN models, we define the *compression ratio* as:

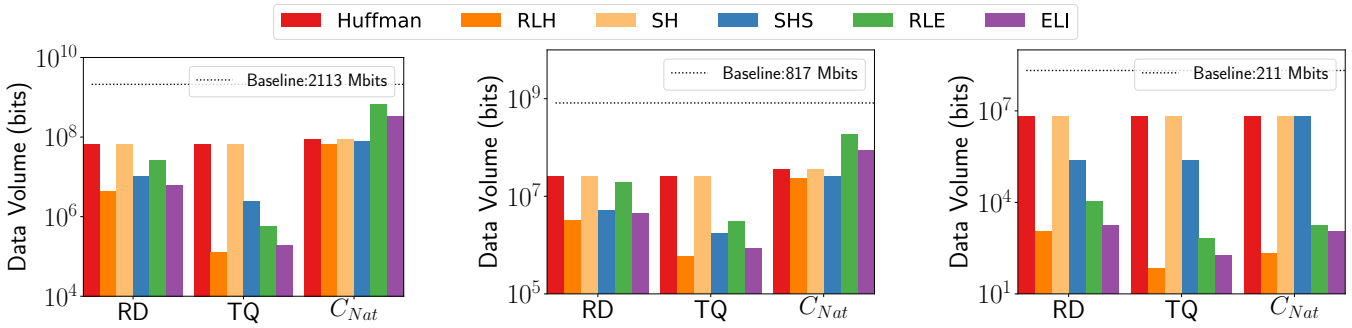
$$\textbf{Compression ratio} := \frac{\text{Bits to represent 32 bit float gradient}}{\text{Bits to represent the encoded gradient}}.$$

We present the best and the second best scores in red and blue colors, respectively. Note that, the code-book length (i.e., decoding information) in most cases are less than 0.1% of the total volume.

Main observations. The baseline is the no-compression run using 32-bit floating-point precision. Since, quantizations are lossy, the accuracy of the training, with qunatization, are less than that of baseline (except for TQ in VGG16 benchmark). In terms of accuracy, C_{Nat} and RD with $s = 256$ are typically able to achieve close baseline model qualities. On the contrary, TQ even with Error-Feedback [42] struggles to achieve acceptable model qualities. This is because the number of quantization states to represent the gradient components of C_{Nat} and RD ($s = 256$) are significantly more compared to TQ which quantizes only the sign of the gradient components. However, the higher quality implies less compression, C_{Nat} and RD ($s = 256$) have the smallest compression-ratios (large volume) while TQ, with 3 states to encode, has the largest compression ratios (small volume).

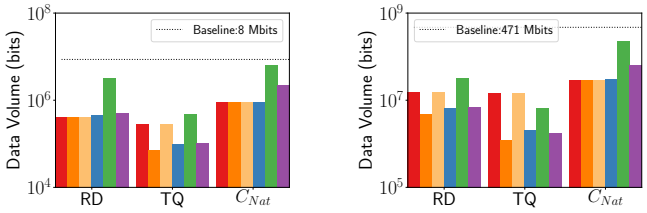
RNN-LSTM results. In Figure 2a, the code-length of most encoders are comparable for C_{Nat}, except for RLE and Elias which results in larger code-length. For both RD and TQ, encoders achieve different code-length and RLH is the smallest among them. Table 3 shows that only RD and C_{Nat} achieves test perplexity close to the no-compression baseline while TQ is not able to converge to the optimal point and has significant gap with the baseline. In terms of volume the encoded compressed gradients of TQ achieves the largest compression ratios. Moreover, RLH is able to reduce the volume over Elias by 81% (or 5.1×).

ResNet-50 results. Figure 2b shows the code-length of the various encoders. In this benchmark, RLH can significantly reduce the length of the code over the other encoders. SHS also performs better



(a) PTB - LSTM (b) ImageNet - ResNet50 (c) ImageNet - GoogleNet

Figure 2: The actual data volume of different methods for different benchmarks.



(a) ResNet20 (b) VGG16

Figure 3: The actual data volume for CIFAR10 benchmark.

than Elias for C_{Nat} and RD. Table 4 shows that only C_{Nat} achieves close accuracy to the no-compression baseline. However, RD and TQ loses about 10 and 25 accuracy points, respectively. This shows the sensitivity of this model, which is based on Residual Networks [19], to compression. In terms of volume, similar to former results, the encoded compressed gradient of TQ achieves the best compression ratios and RLH can reduce the encoded volume over RLE by 88% (or 8.3×) and Elias by 74% (or 3.76×).

GoogLeNet Results. Figure 2c shows the code-length of the encoders. The code-length pattern is different from all the other benchmarks. The code-length of (NH and SH) encoders are the same for all quantiziers. Unlike other benchmarks, the encoders are more space efficient on quantized vectors of C_{Nat} compared to RD. Yet, still TQ achieves the least volume and RLH achieves the least code-length.

ResNet-20 results. In Figure 3a, we present the code-length from various encoders. The code-length of most encoders are comparable for C_{Nat} and RD. In TQ, RLH can significantly reduce the length of the code over the other encoders. Table 4 shows that only RD and C_{Nat} achieves the close accuracy to the no-compression baseline while TQ loses about 7.5 accuracy points. However, in terms of volume, the encoded compressed gradient of TQ achieves the best compression ratios. Moreover, RLH can reduce the volume over Elias by up to 32% (or 1.5×).

VGG-16 results. Figure 3b shows similar pattern for C_{Nat} and TQ but slightly different pattern for RD. RLH can achieve the smallest code-length among all the encoders. Also, SHS can achieve better results compared to Elias. Table 4 shows that TQ achieves in this case the best model quality excelling over the baseline while RD and C_{Nat} achieves lower but close accuracy to the no-compression baseline due to the dense nature of the VGG16 DNN making it resilient to compression. In terms of volume, the encoded compressed gradient of TQ has still the highest compression ratios followed by RD then C_{Nat}. Moreover, RLH can reduce the communicated volume over Elias by up to 55% (or 2.2×).

Table 4: Accuracy and comp. ratios for CNN models

Model	Baseline Accuracy (↑)	Quantizer	Quantized Accuracy	Compression Ratio (×)					
				Huffman	Elias	RLE	SH	SHS	RLH
ResNet50	53.317	RD	43.256	31	184	41	31	156	253
		TQ	25.34	32	947	266	32	470	1353
		C_{Nat}	53.00	23	9	4	23	32	35
ResNet20	94.286	RD	93.333	21	17	2	21	19	21
		TQ	86.786	31	82	18	31	87	121
		C_{Nat}	93.375	10	4	1	10	9	10
VGG16	96.75	RD	94.141	30	68	14	30	71	98
		TQ	97.375	32	265	70	32	231	391
		C_{Nat}	95.125	16	7	2	16	15	16

4.1 Limitation of encoding techniques

The main drawback for quantization techniques is the computational complexity of the associated encoding mechanisms which prohibits their wide-adoption. Our prototype, which uses high-level APIs of the ML framework, is not computationally efficient and is not optimized with respect to the run-time. For this reason, this work only provides empirical-based guidance into the efficiency of different encoders with respect to the communicated volume. However, quantization and encoding can be generally recommended only if their implementations can leverage some-degree of parallelism in the modern hardware accelerators. Recently, commercial and research compression prototypes are leveraging specialized modern hardware (e.g., FPGA [29]).

5 CONCLUSION

In this work, we explore the efficiency of the encoding techniques necessary for communicating quantized gradient vectors during distributed DNN training. We highlight the limitations of different encoders commonly used by many recent quantizers and propose variants of Huffman coding to mitigate them. We analyze the complexities of the encoding techniques and empirically evaluate their performance in popular benchmarks. Our evaluation shows that while the commonly used Elias encoder achieves acceptable compressed volumes, the proposed Huffman-based encoders (RLH, SHS and SH) can excel over Elias-based encoders and provide further reductions by up to 5.1×, 4.32× and 3.8×, respectively. As part of our future work, we aim to implement our schemes on modern accelerators and conduct extensive empirical analysis.

REFERENCES

[1] A. F. Aji and K. Heafield. 2017. Sparse Communication for Distributed Gradient Descent. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. 440–445.

[2] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. 2017. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *NeurIPS*. 1709–1720.

[3] Shashwat Banchhor, Rishikesh Gajjala, Yogish Sabharwal, and Sandeep Sen. 2020. Decode efficient prefix codes. *CoRR* abs/2010.05005 (2020). arXiv:2010.05005 https://arxiv.org/abs/2010.05005

[4] D. Basu, D. Data, C. Karakus, and S. Diggavi. 2019. Qsparse-local-SGD: Distributed SGD with Quantization, Sparsification, and Local Computations. In *NeurIPS*.

[5] R. Bekkerman, M. Bilenko, and J. Langford. 2011. *Scaling up machine learning: Parallel and distributed approaches.* Cambridge University Press.

[6] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar. 2018. SIGNSGD: Compressed Optimisation for Non-Convex Problems. In *International Conference on Machine Learning (ICML)*. 559–568.

[7] Y. Chen, T. Krishna, J. S. Emer, and V. Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

[8] Y. Choi, M. El-Khamy, and J. Lee. 2020. Universal Deep Neural Network Compression. *IEEE Journal of Selected Topics in Signal Processing* (2020), 1–1.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (third ed.). The MIT Press.

[10] Thomas M. Cover and Joy A. Thomas. 2006. *Elements of information theory* (2nd. ed.). Wiley.

[11] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng. 2012. Large Scale Distributed Deep Networks. In *NeurIPS*. 1223–1231.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*.

[13] A. Dutta, E. H. Bergou, A. M. Abdelmoniem, C.-Y. Ho, A. N. Sahu, M. Canini, and P. Kalnis. 2020. On the Discrepancy between the Theoretical Analysis and Practical Implementations of Compressed Communication for Distributed Deep Learning. In *Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)*. 3817–3824.

[14] P. Elias. 1975. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory* 21, 2 (1975), 194–203.

[15] R. Gallager and D. van Voorhis. 1975. Optimal source codes for geometrically distributed integer alphabets (Corresp.). *IEEE Transactions on Information Theory* 21, 2 (March 1975), 228–230. https://doi.org/10.1109/TIT.1975.1055357

[16] Shiming Ge, Zhao Luo, Shengwei Zhao, Xin Jin, and Xiao-Yu Zhang. 2017. Compressing deep neural networks for efficient visual inference. In *IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 667–672.

[17] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

[18] R. Hashemian. 1995. Memory efficient and high-speed search Huffman coding. *IEEE Transactions on Communications* 43, 10 (1995), 2576–2581.

[19] K. He, X. Zhang, S. Ren, and J. Sun. 2015. Deep Residual Learning for Image Recognition. In *CVPR*.

[20] S. Hochreiter and J. Schmidhuber. 1997. Long Short-Term Memory. *Neural Computing* 9, 8 (1997).

[21] Samuel Horváth, Chen-Yu Ho, Ludovit Horvath, Atal Narayan Sahu, Marco Canini, and Peter Richtarik. 2019. Natural Compression for Distributed Deep Learning. *arXiv preprint arXiv:1905.10988* (2019).

[22] D. A. Huffman. 1952. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE* 40, 9 (1952), 1098–1101.

[23] Jiawei Jiang, Fangcheng Fu, Tong Yang, Yingxia Shao, and Bin Cui. 2020. SKCompress: compressing sparse and nonuniform gradient in distributed machine learning. *The VLDB Journal* (2020), 1–28.

[24] H. Kaiming, Z. Xiangyu, R. Shaoqing, and S. Jian. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.

[25] Michael Kohn. 2005. Huffman/CCITT Compression In TIFF. (2005). https://www.mikekohn.net/file_formats/tiff.php

[26] A. Krizhevsky and G. Hinton. 2009. Learning multiple layers of features from tiny images. *Technical report, University of Toronto* 1, 4 (2009).

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NeurIPS*. 1097–1105.

[28] Lawrence L. Larmore and Teresa M. Przytycka. 1995. Constructing Huffman Trees in Parallel. *SIAM J. Comput.* 24, 6 (1995), 1163–1169.

[29] Y. Li, J. Park, M. Alian, Y. Yuan, Z. Qu, P. Pan, R. Wang, A. Schwing, H. Esmaeilzadeh, and N. S. Kim. 2018. A Network-Centric Hardware/Algorithm Co-Design to Accelerate Distributed Training of Deep Neural Networks. In *IEEE/ACM International Symposium on Micro-architecture (MICRO)*. 175–188.

[30] Y. Lin, S. Han, H. Mao, Y. Wang, and W. Dally. 2018. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. In *International conference on Learning Representation (ICLR)*.

[31] M. P. Marcus, B. Santorini, M. A. Marcinkiewicz, and A. Taylor. 1999. Treebank-3. (1999). https://catalog.ldc.upenn.edu/LDC99T42.

[32] A. Moffat and A. Turpin. 1997. On the implementation of minimum redundancy prefix codes. *IEEE Transactions on Communications* 45, 10 (1997), 1200–1207.

[33] R. A. Patel, Y. Zhang, J. Mak, A. Davidson, and J. D. Owens. 2012. Parallel lossless data compression on the GPU. In *Innovative Parallel Computing (InPar)*. 1–9.

[34] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. 2019. A Generic Communication Scheduler for Distributed DNN Training Acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*.

[35] Pytorch.org. 2019. PyTorch. (2019). https://pytorch.org/

[36] A. H. Robinson and C. Cherry. 1967. Results of a prototype television bandwidth compression scheme. *Proc. IEEE* 55, 3 (1967), 356–364.

[37] F. Sattler, Simon Wiedemann, K-R Müller, and W. Samek. 2019. Sparse Binary Compression: Towards Distributed Deep Learning with minimal Communication. In *International Joint Conference on Neural Networks, IJCNN*. 1–8.

[38] Benjamin Schlegel, Rainer Gemulla, and Wolfgang Lehner. 2010. Fast Integer Compression Using SIMD Instructions. In *Proceedings of the Sixth International Workshop on Data Management on New Hardware (DaMoN)*.

[39] Jürgen Schmidhuber and Stefan Heil. 1995. Predictive coding with neural nets: Application to text compression. In *NeurIPS*. 1047–1054.

[40] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[41] K. Simonyan and A. Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*.

[42] S. U. Stich, J. B. Cordonnier, and M. Jaggi. 2018. Sparsified SGD with memory. In *NeurIPS*. 4447–4458.

[43] N. Strom. 2015. Scalable distributed DNN training using commodity GPU cloud computing. In *INTERSPEECH*. 1488–1492.

[44] B. Sukhwani, B. Abali, B. Brezzo, and S. Asaad. 2011. High-Throughput, Lossless Data Compresion on FPGAs. In *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*. 113–116.

[45] H. Sun, Y. Shao, J. Jiang, B. Cui, K. Lei, Y. Xu, and J. Wang. 2019. Sparse Gradient Compression for Distributed SGD. In *Database Systems for Advanced Applications*. 139–155.

[46] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. 2015. Going Deeper with Convolutions. In *Computer Vision and Pattern Recognition (CVPR)*. 1–9.

[47] T. Vogels, S. P. Karimireddy, and M. Jaggi. 2019. PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization. *NeurIPS*.

[48] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright. 2018. ATOMO: Communication-efficient Learning via Atomic Sparsification. In *NeurIPS*. 9850–9861.

[49] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li. 2017. TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning. In *NeurIPS*. 1508–1518.

[50] H. Xu, C.-Y. Ho, A. M. Abdelmoniem, A. Dutta, E. H. Bergou, K. Karatsenidis, M. Canini, and P. Kalnis. 2020. *Compressed Communication for Distributed Deep Learning: Survey and Quantitative Evaluation.* Technical Report. KAUST. http://hdl.handle.net/10754/631179.

[51] Yue Yu, Jiaxiang Wu, and Junzhou Huang. 2019. Exploring Fast and Communication-Efficient Algorithms in Large-Scale Distributed Networks. In *AISTATS*.