Compressing Deep Neural Networks with Probabilistic Data Structures

Brandon Reagen
Harvard University

Udit Gupta
Harvard University

Robert Adolf
Harvard University

Michael M. Mitzenmacher
Harvard University

Alexander M. Rush
Harvard University

Gu-Yeon Wei
Harvard University

David Brooks
Harvard University/Facebook

ABSTRACT

This paper presents a lossy weight encoding method which complements conventional compression techniques including weight pruning and clustering. The encoding is based on the Bloomier filter, a probabilistic data structure that can save space at the expense of introducing random errors in the weights. Leveraging the ability of DNNs to tolerate these imperfections and by re-training around them, the proposed technique can compress DNN weights by up to 496× (a 1.51× improvement over the state-of-the-art) without sacrificing model accuracy.

1 INTRODUCTION

The continued success of deep neural networks (DNNs) comes with increasing demands on compute, memory, and networking resources. Moreover, the correlation between model size and accuracy suggests that tomorrow's networks will only grow larger. This growth presents a challenge for resource-constrained platforms such as mobile phones and wireless sensors. As new hardware now enables executing DNN inference on edge devices [1, 10], a practical issue is distributing the latest models, especially in regions not equipped with high-bandwidth networks. For instance, it is estimated that, globally, 800 million users will still use 2G networks in 2020 [4], which can take up to 30 minutes to download 20MB of data. Today's DNNs are on the order of tens to hundreds of MBs, making them difficult to distribute even on high-speed connections. In order to support state-of-the-art deep learning methods on edge devices, techniques to reduce the size of DNN models without sacrificing model accuracy are needed.

Model compression is a popular solution for this problem. A variety of compression algorithms have been proposed in recent years, and many exploit the intrinsic redundancy in model weights. The majority of this work has focused on ways of simplifying or eliminating weight values (e.g., through weight pruning and quantization), while comparatively little effort has been spent on devising techniques for encoding and compressing.

In this paper we present Weightless [11]: a novel lossy encoding method based on Bloomier filters, a probabilistic data structure [3]. Bloomier filters inexactly store a function map, and by adjusting the filter parameter, we can elect to use less space at the cost of erroneous values.

We use this data structure to compactly encode the weights of a DNN, exploiting redundancy in the weights to tolerate some errors. In conjunction with existing weight simplification techniques, namely pruning and clustering, our approach dramatically reduces memory and bandwidth requirements for over-the-wire transmission of DNNs. Weightless demonstrates compression rates of up to 496× without loss of accuracy, improving on the state of the art by up to 1.51×.

2 WEIGHTLESS

2.1 The Bloomier filter

A Bloomier filter generalizes the idea of a Bloom filter [2]. Given a subset S of a universe U, a Bloom filter answers queries of the form, "Is $v \in S$?". If v is in S, the answer is always yes; if v is not in S, there is some probability of a false positive. By allowing false positives, Bloom filters can dramatically reduce the space needed to represent the set. A Bloomier filter [3] is a similar data structure but instead encodes a function. For each v in a domain S, the function has an associated value $f(v)$ in the range $R = [0, 2^r)$. Given an input v, a Bloomier filter always returns $f(v)$ when v is in S. When v is not in S, the Bloomier filter returns a null value $\perp$, except that some fraction of the time there is a "false positive", and the Bloomier filter returns an incorrect, non-null value in the range R.

Decoding Let S be the subset of values in U to store, with $|S| = n$. A Bloomier filter uses a small number of hash functions (typically four), and a hash table $\mathbf{X}$ of $m = cn$ cells for some constant c (1.25 in this paper), each holding $t > r$ bits. For hash functions H_0, H_1, H_2, H_M, let $H_{0,1,2}(v) \rightarrow [0, m)$ and $H_M(v) \rightarrow [0, 2^r)$, for any $v \in U$. The table $\mathbf{X}$ is set up such that for every $v \in S$,

$$X_{H_0(v)} \oplus X_{H_1(v)} \oplus X_{H_2(v)} \oplus H_M(v) = f(v).$$

Hence, to find the value of $f(v)$, hash v four times, perform three table lookups, and exclusive-or together the four values. Like the Bloom filter, querying a Bloomier filter runs in $O(1)$ time. For $u \notin S$, the result, $X_{H_0(u)} \oplus X_{H_1(u)} \oplus X_{H_2(u)} \oplus H_M(u)$, will be uniform over all t-bit values. If this result is not in $[0, 2^r)$, then $\perp$ is returned, and if it happens to land in $[0, 2^r)$, a false positive occurs and a non-$\perp$ result is (incorrectly) returned. An incorrect value is therefore returned with probability 2^{r-t}.

2.2 Weight encoding with Bloomier filters

We propose using the Bloomier filter to compactly store weights in a DNN. The function f encodes the mapping between indices of nonzero weights to their corresponding values. Given a weight

Please use [11] to reference Weightless.

matrix **W**, define the domain S to be the set of indices $\{i, j \mid w_{i,j} \neq 0\}$. Likewise, the range R is $[-2^{a-1}, 2^{a-1}) - \{0\}$ for a such that all values of **W** fall within the interval. To enable weight value clustering (see below) this range is remapped to $[0, 2^r)$ and encodes the cluster indices. A null filter response means the weight value is zero.

Once f is encoded in a filter, an approximation $\mathbf{W}'$ of the original weight matrix is reconstructed by querying all indices. The original nonzero elements of **W** are preserved in the approximation, as are most of the zero elements. A small subset of zero-valued weights in $\mathbf{W}'$ will take on nonzero values as a result of random collisions in **X**, possibly changing the model's output. As DNNs are well known to be robust to weight errors (e.g., [12]), we suspected the resulting false positives would not significantly impact model accuracy. The relationship between false positives and accuracy is given in [11].

Complementing Bloomier filters with simplification Because the space used by a Bloomier filter is $O(nt)$, they are especially useful under two conditions: (1) The stored function is sparse (small n, with respect to $|U|$), and (2) It has a restricted range of output values (small r, since $t > r$). To improve overall compression, we pair approximate encoding with weight transformations.

Pruning networks to enforce sparsity (condition 1) has been studied extensively [7, 8]. Weightless considers two different pruning techniques. Magnitude pruning with retraining is straightforward to use and offers good results. DNS [5] is a more aggressive technique proposed recently that prunes the network during training. We were able to acquire two sets of models, LeNet-300-100 and LeNet5, that were pruned using DNS and include them in our evaluation (VGG-16 is not available). Improving sparsity reduces the overall encoding size linearly with the number of non-zeros with no effect on the false positive rate (which depends only on r and t). We use two methods to demonstrate the benefits of Weightless as networks increase in sparsity, since the DNS networks are notably more sparse than the same magnitude pruned networks.

Reducing r (condition 2) amounts to minimizing the number of bits required to represent weight values. While many solutions to discretize weights exist, we use k-means clustering. After clustering the weights, the k centroids are saved into an auxiliary table and the elements of **W** are replaced with indices into this table. This style of indirect encoding is especially well-suited to Bloomier filters, as these indices come from a small, contiguous set of integers.

Tuning the t hyperparameter Bloomier filters introduce an additional hyperparameter t (the number of bits per cell). t trades off the Bloomier filter's size and the false positive rate which effects model accuracy. While t needs to be tuned, we find it far easier to reason about than most other DNN hyperparameters. Because we encode k clusters, t must be greater than $\lceil \log_2 k \rceil$, and each additional t bit reduces the number of false positives by a factor of 2. This limits the number of reasonable values for t: networks experience substantial accuracy loss when t is too low, but high values of t are wasteful because DNNs have enough implicit resilience to handle some errors. Experimentally we find that t typically falls in the range of 6 to 9 for our models.

Retraining to mitigate the effects of false positives We encode each layer's weights sequentially. Because the weights are fixed, the Bloomier filter's false positives are deterministic. This allows for the retraining of deeper network layers to compensate for errors. It is important to note that encoded layers are not retrained. If an encoded layer were retrained, a new encoding would have to be constructed (because S changes) and the indices of false positives would differ after every iteration of retraining. Instead, we find retraining all subsequent layers to be sufficient and effective, typically allowing us to reduce t by one or two bits.

Table 1: Weightless results on the largest layers of common models used to evaluate compression. Results are compared to the current state-of-the-art method [6] (Huffman below). The improvement (Improve) column shows Weightless offers up to a 1.51× improvement over [6].

Model	Pruning Method	Layer	Compression Factor		
			Huffman	Weightless	Improve
LeNet-FC	Magnitude	FC-0	59.1×	60.1×	1.02×
		FC-1	56.0×	64.3×	1.15×
	DNS	FC-0	153×	174×	1.13×
		FC-1	129×	195×	**1.51**×
LeNet5	Magnitude	CNN-1	42.8×	51.6×	1.21×
		FC-0	59.1×	74.2×	1.25×
	DNS	CNN-1	89.5×	114.4×	1.28×
		FC-0	333×	**496**×	1.49×
VGG-16	Magnitude	FC-0	119×	157×	1.31×
		FC-1	88.4×	85.8×	0.97×

Compressing Bloomier filters When sending weight matrices over a network, it is not necessary to retain the ability to access weight values as they are being sent, so it is advantageous to add another layer of compression for transmission. We use arithmetic coding, an entropy-optimal stream code which exploits the distribution of values in the table [9].

3 RESULTS AND FUTURE DIRECTIONS

We evaluate Weightless using three models commonly used in DNN compression papers: LeNet-300-100 (LeNet-FC here), LeNet5, and VGG-16. Table 1 shows the results of applying Weightless to the largest layers of these models. Hyperparameter settings for the results can be found in [11]. The table includes compression ratios relative to 32-bit, non-pruned models. Compared to this baseline, Weightless is able to reduce the memory footprint by up to 496×. We also compare results to the current accepted weight compression method [6]. Weightless offers up to a 1.51× improvement over this aggressive baseline (CSR and Huffman encoding).

Looking forwards, we see avenues for continuing this line of research. First, as better mechanisms for pruning model weights are discovered, end-to-end compression with Weightless will improve commensurately. Weightless also scales much better with sparsity than existing methods [11]. Second, the theory community has already developed more advanced—albeit more complicated—construction algorithms for Bloomier filters, which promise asymptotically better space utilization compared to the method used in this paper. Finally, by demonstrating the opportunity for using lossy encoding schemes for model compression, we hope we have opened the door for more research on encoding algorithms and novel uses of probabilistic data structures.

REFERENCES

[1] Apple. 2017. The future is here: iPhone X. https://www.apple.com/newsroom/2017/09/the-future-is-here-iphone-x/. (2017).

[2] Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426.

[3] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. 2004. The Bloomier filter: an efficient data structure for static support lookup tables. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms.*

[4] GSMA. 2014. Half of the WorldâĂŹs Population Connected to the Mobile Internet by 2020, According to New GSMA Figures. https://www.gsma.com/newsroom/press-release/. (Novemeber 2014).

[5] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic Network Surgery for Efficient DNNs. In *Advances in Neural Information Processing Systems 29.*

[6] Song Han, Huizi Mao, and Bill Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations.*

[7] Babak Hassibi and David G Stork. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 6.*

[8] Yann LeCun, John S. Denker, and Sara A. Solla. 1989. Optimal Brain Damage. In *Advances in Neural Information Processing Systems 2.*

[9] David J.C. MacKay. 2005. *Information Theory, Inference, and Learning Algorithms* (fourth printing ed.). Cambridge University Press.

[10] Qualcomm. 2017. Snapdragon Neural Processing Engine Now Available on Qualcomm Developer Network. https://www.qualcomm.com/news/releases/2017/07/25/snapdragon-neural-processing-engine-now-available-qualcomm-developer/. (2017).

[11] Brandon Reagen, Udit Gupta, Robert Adolf, Michael M. Mitzenmacher, Alexander M. Rush, Gu-Yeon Wei, and David M. Brooks. 2017. Weightless: Lossy Weight Encoding For Deep Neural Network Compression. *CoRR* abs/1711.04686 (2017). arXiv:1711.04686 http://arxiv.org/abs/1711.04686

[12] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators. In *Proceedings of the 43rd International Symposium on Computer Architecture.*