Succinct Source Coding of Deep Neural Networks

Sourya Basu and Lav R. Varshney
Department of Electrical and Computer Engineering, Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
{sourya, varshney}@illinois.edu

Abstract

Deep neural networks have shown incredible performance for inference tasks in a variety of domains. Unfortunately, most current deep networks are enormous cloud-based structures that require significant storage space, which limits scaling of deep learning as a service (DLaaS) and use for on-device augmented intelligence. This paper finds algorithms that directly use lossless compressed representations of deep feedforward networks (with synaptic weights drawn from discrete sets), to perform inference without full decompression. The basic insight that allows less rate than naïve approaches is the recognition that the bipartite graph layers of feedforward networks have a kind of permutation invariance to the labeling of nodes, in terms of inferential operation and that the inference operation depends locally on the edges directly connected to it. We also provide experimental results of our approach on the MNIST dataset.

1 Introduction

Deep learning has achieved incredible performance for inference tasks such as speech recognition, image recognition, and natural language processing. Most current deep neural networks, however, are enormous cloud-based structures that are *too large* and *too complex* to perform fast, energy-efficient inference on device or for scaling deep learning as a service (DLaaS). Compression, with the capability of providing inference without full decompression, is important. Universal source coding for feedforward deep networks having synaptic weights drawn from finite sets that essentially achieve the entropy lower bound were introduced in (1). Here, we provide—for the first time—an algorithm that directly uses these compressed representations for inference tasks without complete decompression. Structures that can represent information near the entropy bound while also allowing efficient operations on them are called *succinct structures* (2; 3; 4). Thus, we provide a succinct structure for feedforward neural networks, which may fit on-device and enable scaling of DLaaS.

Related Work: There has been recent interest in compact representations of neural networks (5; 6; 7; 8; 9; 10; 11; 12; 13; 14). While most of these algorithms are lossy, we provide an efficient lossless algorithm, which can be used on top of any lossy algorithm that quantizes or prunes network weights; prior work on lossless compression of neural networks either used Huffman coding in a way that did not exploit invariances or was not succinct and required full decompression for inference. The proposed algorithm builds on the sublinear entropy-achieving representation in (1) but is the first time succinctness—the further ability to perform inference with negligible space needed for partial decompression—has been attempted or achieved. Our inference algorithm is similar to arithmetic decoding and so computational performance is also governed by efficient implementations of arithmetic coding. Efficient high-throughput implementations of arithmetic coding/decoding have been developed for video, e.g. as part of the H.264/AVC and HEVC standards (15; 16).

2 Feedforward neural network structure

Let us describe the neural network model considered in (1) which will be used here to develop succinct structures of deep neural networks. In a feedforward neural network, each node j com-

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

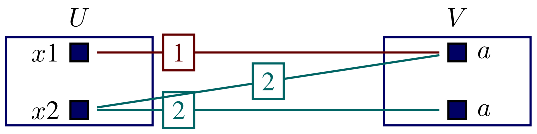

Figure 1: Partially-labeled bipartite graph with edge colors $\{0, 1, 2\}$, where an edge of color 0 between a vertex from U and a vertex from V indicates they are not connected.

putes an activation function $g(\cdot)$ applied to the weighted sum of its inputs, which we can note is a permutation-invariant function: $a_j = g\left(\sum_i w_{ij}a_i\right) = g\left(\sum_i w_{\pi(i)j}a_{\pi(i)}\right)$, for any permutation π. Consider a feedforward neural network with $K-1$ hidden layers where each node contains N nodes (for notational convenience) such that the nodes in all the $K-1$ hidden layers are indistinguishable from each other (when edges are ignored) but the nodes in the input and output layers are labeled and can be distinguished. There is an edge of color i, $i = 0, \ldots, m$, between any two nodes from two different layers independently with probability p_i, where p_0 is the probability of no edge. Consider a substructure: *partially-labeled bipartite graphs*, see Fig. 1, which consists of two sets of vertices containing N vertices each with one of the sets containing labeled vertices and the other set containing unlabeled vertices. An edge of color i exists between any two nodes taken one from each set with probability p_i, $i = 0, \ldots, m$ where p_0 is the probability of no edge. Refer to (1) for detailed discussion on the structure. To construct the K-layer neural network, think of it as made of a partially-labeled bipartite graph for the first two layers but then each time the nodes of an unlabeled layer are connected, we treat it as a labeled layer, based on its connection to the previous labeled layer (i.e. we can label the unlabeled nodes based on the nodes of the previous layer it is connected to), and iteratively complete the K-layer neural network.

3 Succinct representation of a partially-labeled bipartite graph

First, we consider the succinct representation of a partially labeled bipartite graph, followed by that of a K-layered neural network. Alg. 1 is an inference algorithm for a partially-labeled bipartite graph with input to the graph X and output Y. Later we use this algorithm to make inferences in a K-layered neural network where outputs of unlabeled layers correspond to outputs of a hidden layer. The optimally compressed representation of a partially-labeled bipartite graph produced by (1, Alg.1) is taken as an input by Alg. 1, in addition to the input X to the graph, and the output Y of the graph is given out. If the graph has N nodes in each layer, then only an additional $O(N)$ bits of dynamic space is required by Alg. 1 for the inference task while it takes $O(N^2)$ bits to store the representation and hence the structure in succinct as discussed below.

Lemma 1. *Output Y obtained from Alg. 1 is a permutation of $\tilde{Y}$, the output from the uncompressed neural network representation.*

Proof. Say, we have an $m \times 1$ vector X to be multiplied with an $m \times n$ weight matrix W, to get the output $\tilde{Y}$, an $n \times 1$ vector. Then, $\tilde{Y} = W^T X$, and so the jth element of $\tilde{Y}$, $\tilde{Y}_j = \sum_{i=1}^{m} W_{j,i}^T x_i$. In Alg. 1, while traversing a particular depth i, we multiply all Y_js with $X_i W_{i,j}$ and hence when we reach depth N, we get the Y vector as required. The change in permutation of $\tilde{Y}$ with respect to Y is because while compressing W, we do not encode the permutation of the columns, retaining the row permutation. □

Theorem 2. *The additional dynamic space requirement of Alg. 1 is $O(N)$, and hence the compressed representation formed in (1, Alg. 1) is succinct.*

Proof. The major dynamic space requirement is for decoding of individual nodes, and the queue, Q. Clearly, the space required for Q, is much more than the space required for decoding a single node. We show the expected space complexity corresponding to Q is less than or equal to $2(m+1)N(1 + 2\log_2(\frac{m+2}{m+1}))$ using Elias-Gamma integer codes for each entry in Q. Note that Q has nodes from at most two consecutive depths, and since only the child nodes of non-zero nodes are encoded, and the number of non-zero nodes at any depth is less than N, we can have a maximum of $2(m+1)N$ nodes encoded in Q. Let $\alpha_0, ..., \alpha_k$ be the non-zero tree nodes at some depth d of the tree, where $k = (m+1)N$. Let S be the total space required to store Q. Using integer codes, we can encode any positive number x in $2\log_2(x) + 1$ bits, and to allow 0, we need $2\log_2(x+1) + 1$ bits. Thus, the

Algorithm 1 Inference algorithm for compressed network.

```
1: Input: $X = [x_1, x_2, \dots, x_N]$, the input to the neural network, and $\mathfrak{L}$, the compressed representation of the partially-labeled bipartite graph obtained from (1, Alg. 1).
2: Output: $Y = [y_1, y_2, \dots, y_N]$, the output vector of the neural network, and $\mathfrak{L}$, the compressed representation as obtained from input.
3: Initialize: $Y = [0, 0, \dots, 0]$, $d = 1$, the number of neurons covered at the current depth, $j = 1$, an empty queue $Q$, and an empty string $\mathfrak{L}_1$ which would return the compressed representation $\mathfrak{L}$ once the algorithm has executed. Enqueue $Q$ with $N$.
4: while $Q$ is not empty and $d \leq N$ do
5:    Set $i = 0$. Set $f$ = the first element obtained after dequeuing $Q$.
6:    while $i \leq m$ and $f > 0$ do
7:       decode the child node of $f$ corresponding to color $i$ and store it as $c$.
8:       Encode $c$ back in $\mathfrak{L}_1$. Enqueue $c$ in $Q$.
9:       Add $x_l \times w_i$ to each of $y_j$ to $y_{(j+c)}$. Add $c$ to $j$.
10:      if $j = 1$, at least one non-zero node has been processed at the current depth then
11:         $d = d + 1$
12:      end if
13:   end while
14: end while
15: Update the $Y$ vector using the required activation function.
```

Table 1: Experimental results for (1, Alg. 1) and Alg. 1

Dimension of weight matrix ($M \times N$)	$MNH(p) - N\log_2 N$	Observed length (bits)	Avg. queue length (bits)	Max. queue length (bits)
$M = 784, N = 50$	188335	188374	151	409
$M = 50, N = 50$	11747	11702	177	476
$M = 50, N = 50$	11747	11945	170	434
$M = 50, N = 50$	11747	11940	164	401

arithmetic-geometric inequality implies $S \leq 2(\sum 2\log_2(\alpha_i + 1) + 1) \leq 2N(m+1) + 4N(m+1)\log_2(\frac{m+2}{m+1})$. From Prop. 1 and (1, Theorem 1), the additional dynamic space for Alg. 1 is $O(N)$, while the entropy of a partially-labeled bipartite graph is $O(N^2)$. Thus, the structure is succinct. □

Now consider the structure of the K-layered neural network as in Sec. 2 and provide its succinct representation. The extra dynamic space for K-layers remains the same as for 2-layers as described in Alg. 1 as inference is done one layer at a time.

Theorem 3. *The compressed structure obtained by the iterative use of (1, Alg. 1) is succinct.*

4 Experiments

We trained a feedforward neural network of dimension $784 \times 50 \times 50 \times 50 \times 50 \times 10$ on the MNIST dataset using gradient descent algorithm to get 98.4% accuracy on the test data. Network weights were quantized using a uniform quantizer into 33 steps to get a network with an accuracy of 97.5% on the training data and an accuracy of 93.48% on the test data. The weight matrices from the second to the last layer were rearranged based on the weight matrices corresponding to the previous layers as needed for Alg. 1 to work. These matrices, except the last matrix connected to the output, were compressed using (1, Alg. 1) to get the compressed network, and arithmetic coding was implemented by modification of an existing implementation. The compressed network performs exactly as the quantized network as it should, since we compress losslessly. We observe that the extra memory required for inference is negligible compared to the size of the compressed network. Detailed results from the experiment and dynamic space requirements are described in Table 1, where $H(p)$ is the empirical entropy calculated from the weight matrices.[1]

[1] Implementations can be found at https://github.com/basusourya/DNN.

References

[1] S. Basu and L. R. Varshney, "Universal source coding of deep neural networks," in *Proc. IEEE Data Compression Conf. (DCC 2017)*, Apr. 2017, pp. 310–319.

[2] R. Raman, V. Raman, and S. S. Rao, "Succinct indexable dictionaries with applications to encoding k-ary trees and multisets," in *Proc. 13th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA'02)*, Jan. 2002, pp. 233–242.

[3] G. J. Jacobson, "Succinct static data structures," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, Jan. 1989.

[4] M. Patrascu, "Succincter," in *Proc. 27th Annu. Symp. Found. Comput. Sci.*, Oct. 2008, pp. 305–313.

[5] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," arXiv:1412.6115 [cs.CV]., Dec. 2014.

[6] M. Courbariaux, Y. Bengio, and J.-P. David, "Low precision arithmetic for deep learning," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, May 2015.

[7] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML 2015)*, Jul. 2015, pp. 1737–1746.

[8] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML 2015)*, Jul. 2015, pp. 2285–2294.

[9] Z. Lu, V. Sindhwani, and T. N. Sainath, "Learning compact recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP 2016)*, Mar. 2016, pp. 5960–5964.

[10] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, May 2016.

[11] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, May 2016.

[12] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, May 2016.

[13] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "DeepX: A software accelerator for low-power deep learning inference on mobile devices," in *Proc. 15th ACM/IEEE Int. Conf. Inf. Processing Sensor Netw. (IPSN)*, Apr. 2016.

[14] A. Chatterjee and L. R. Varshney, "Towards optimal quantization of neural networks," in *Proc. 2017 IEEE Int. Symp. Inf. Theory*, Jun. 2017, pp. 1162–1166.

[15] V. Sze and M. Budagavi, "High throughput CABAC entropy coding in HEVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1778–1791, Dec. 2012.

[16] V. Sze and D. Marpe, "Entropy coding in HEVC," in *High Efficiency Video Coding (HEVC)*, V. Sze, M. Budagavi, and G. J. Sullivan, Eds. Springer, 2014, pp. 209–274.