UNIVERSAL AND SUCCINCT SOURCE CODING OF DEEP NEURAL NETWORKS

BY

SOURYA BASU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Adviser:

Assistant Professor Lav R. Varshney

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 intelligence. This work is concerned with finding universal lossless compressed representations of deep feedforward networks with synaptic weights drawn from discrete sets, and directly performing inference without full decompression. The basic insight that allows less rate than naïve approaches is recognizing that the bipartite graph layers of feedforward networks have a kind of permutation invariance to the labeling of nodes, in terms of inferential operation. We provide efficient algorithms to dissipate this irrelevant uncertainty and then use arithmetic coding to nearly achieve the entropy bound in a universal manner. We also provide experimental results of our approach on several standard datasets.

To my family and friends, for their love and support.

ACKNOWLEDGMENTS

I would like to thank my adviser Prof. Lav R. Varshney for his guidance and support throughout this work on information-theoretic compression of deep neural networks. I am really grateful to him for sharing a number of interesting ideas and helping me develop them. I would also like to thank my group and labmates Haizi Yu, Houzhi Zhou, Vishal Rana, Aditya Deshmukh, Srilakshmi Pattabiraman, Akshayaa Magesh, Sam Spencer, and Alan Yang, and former members Ravi Kiran Raman, Daewon Seo, Xiou Ge, Sakshi Agarwal, Rebecca Chen, and Ruby Zhuang for a number of interesting discussions.

This work was supported in part by Systems on Nanoscale Information fabriCs (SONIC), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA, and in part by grant number 2018-182794 from the Chan Zuckerberg Initiative DAF, an advised fund of Silicon Valley Community Foundation.

TABLE OF CONTENTS

CHAPTER 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. Even in the cloud, providing personalized deep learning as a service (DLaaS), where each customer for an application like bank fraud detection may require a different trained network, scaling to millions of stored networks is not possible. Compression, with the capability of providing inference without full decompression, is important. We develop new universal source coding techniques for feedforward deep networks having synaptic weights drawn from finite sets that essentially achieve the entropy lower bound, which we also compute. Further, we provide an algorithm to use 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* [1–4]. Thus, we provide a succinct structure for feedforward neural networks, which may fit on-device and may enable scaling of DLaaS in the cloud.

Over the past couple of years, there has been growing interest in compact representations of neural networks [5–14], largely focused on lossy representations, see [15] for a recent survey of developed techniques including pruning, pooling, and factoring. These works largely lack strong information-theoretic foundations and may discretize real-valued weights through simple uniform quantization, perhaps followed by independent entropy coding applied to each. It is worth noting that binary-valued neural networks (having only a network structure [16] rather than trained synaptic weights) can often achieve high-fidelity inference [17, 18] and that there is a view in neuroscience that biological synapses may be discrete-valued [19].

Neural networks are composed of nodes connected by directed edges. Feedforward networks (multilayer perceptrons) have connections in one direction, arranged in layers. An edge from node i to node j propagates an activation value a_i from i to j, and each edge has a synaptic weight w_{ij} that determines the sign/strength of the connection. Each node j computes an activation function $g(\cdot)$ applied to the weighted sum of its inputs, which we can note is a permutation-invariant function:

$$a_j = g\left(\sum_i w_{ij} a_i\right) = g\left(\sum_i w_{\pi(i)j} a_{\pi(i)}\right) \tag{1.1}$$

for any permutation π. Nodes in the second layer are indistinguishable.

Taking advantage of this permutation invariance in the structure of neural networks (previously unrecognized, e.g. [20]) for lossless entropy coding can lead to rate reductions on top of any lossy representation technique that has been developed [15]. In particular, the structure of feedforward deep networks in layers past the input layer are unlabeled bipartite graphs where node labeling is irrelevant, much like for nonsequential data [21–23]. By dissipating the uncertainty in this invariance, lossless coding can compress more than universal graph compression for labeled graphs [24], essentially a gain of $N \log N$ bits for networks with N nodes.

The first main contribution of this work is determining the entropy limits, once the appropriate invariances are recognized. Next, to design an appropriate "sorting" of synaptic weights to put them into a canonical order where irrelevant uncertainty due to invariance is removed; a form of arithmetic coding is then used to represent the weights [25, 26]. Note that the coding algorithm essentially achieves the entropy bound. The third main contribution is an efficient inference algorithm that uses the compressed form of the feedforward neural network to calculate its output without completely decoding it, taking only $O(N)$ additional dynamic space for a network with N nodes in the layer with maximum number of nodes. Finally, the work provides experimental results of our compression and inference algorithms on feedforward neural networks trained to perform classification tasks on standard MNIST, IMDB, and Reuters datasets.

1.1 Overview

In this section, we describe the flow of the work. In Ch. 2, we discuss the basic structure and invariant properties of a feedforward neural network (multilayer perceptron), and how it can be decomposed into substructures that we call partially labeled bipartite graphs and unlabeled bipartite graphs. In Ch. 3 and Ch. 4, we provide entropy bounds, universal compression algorithms, and inference algorithms that need not require full decompression for both partially labeled bipartite graphs and unlabeled bipartite graphs as defined in Ch. 2, respectively. Chapter 5 provides two different compression algorithms based on the compression algorithms provided in Ch. 3 and Ch. 4 respectively. Chapter 5 also provides an efficient inference algorithm based on the inference algorithm provided in Ch. 3 that makes use of the compressed feedforward neural network for inference without fully decompressing it. Chapter 6 provides experimental results for the compression algorithms and Ch. 7 concludes the work.

1.2 Bibliographical Note

The work on universal compression of densely connected neural networks was presented in

- S. Basu and L. R. Varshney, "Universal Source Coding of Deep Neural Networks," in *Proceedings of the IEEE Data Compression Conference*, Snowbird, Utah, 4-7 April 2017.

The work on succinctness was presented in

- S. Basu and L. R. Varshney, "Succinct Source Coding of Deep Neural Networks," in *Proceedings of NeurIPS Compact Deep Neural Network Representation with Industrial Applications Workshop (CDNNRIA)*, Montreal, Canada, 7 December 2018.

And the complete work dealing with both universal compression that is also succinct was presented at

- S. Basu and L. R. Varshney, "Universal and Succinct Source Coding of Deep Neural Networks," presented at *Stanford Compression Workshop*, Palo Alto, California, 15 February 2019.

A preprint for this work is available on ArXiv

- S. Basu and L. R. Varshney, "Universal and Succinct Source Coding of Deep Neural Networks," arXiv: 1804.02800 [cs.IT].

CHAPTER 2

FEEDFORWARD NEURAL NETWORK STRUCTURES

Feedforward neural network forms a very important class of neural network structures where the connections between the nodes do not form a cycle. Examples of such structures include densely connected neural networks, convolutional neural networks (CNNs) [27], and group equivariant convolutional neural networks (GCNNs) [28–30]. In densely connected neural networks every node in a layer in the network is connected to every node in the subsequent layer whereas in convolutional neural networks the connections between consecutive layers are in the form of convolution of filters which provides translational equivariance making such neural networks extremely useful in the domain of image processing. GCNNs is a generalization of CNNs which form convolutional connections between layers just like CNNs, however, these networks are equivariant to more general form of transformations than in CNNs which is possible because of use of specially designed filters. Another example of a feedforward neural network is given by [31], that uses PR product instead of inner product for computation to help improve the performance of several state-of-the-art classification networks. Although, these networks are a different form of computation, their structure remains the same as that of densely connected feedforward neural networks. Hence, compression of these networks are the same as densely connected neural networks modeled in Sec. 2.1.

In Sec. 2.1, we discuss a probabilistic model for densely connected feedforward neural networks that we use for our work. In the subsequent sections, we provide a discussion on CNNs, and GCNNs indicating the possible extension of the compression algorithms in our work to such architecture of neural networks.

2.1 Dense Feedforward Neural Network Structures

Consider a K-layer feedforward neural network with each (for notational convenience) layer having N nodes, such that nodes in the first layer are labeled and all nodes in each of the remaining $(K-1)$ layers are indistinguishable from each other (when edges are ignored) due to the inferential invariance discussed in (1.1). Suppose there are m possible colorings of edges (corresponding to synaptic weights), and that connections from each node in a layer to any given node in the next layer takes color i with probability p_i, $i = 0, \ldots, m$, where p_0 is the probability of no edge. The goal is to universally find an efficient representation of this neural network structure. We will first consider optimal representation for two smaller substructures that form the layers of feedforward neural networks (after recognizing the invariance), and then return to the problem of optimally representing the full network. The problem of neural network inference without the need to decode is interspersed in describing representations for the substructures and the full network (in Ch. 3 and Ch. 5, we consider the problem of inference without the need to decode for partially labeled bipartite graphs and feedforward neural networks respectively).

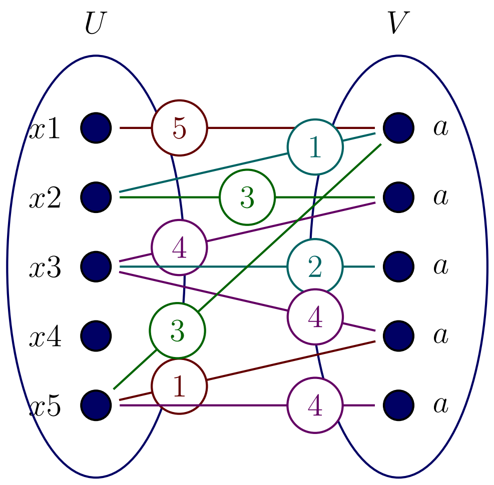
Let us define the two aforementioned substructures: *partially labeled bipartite graphs* and *unlabeled bipartite graphs*, see Fig. 2.1.

Definition 1. A *partially labeled bipartite graph* consists of two sets of vertices, U and V. The set U contains N labeled vertices, whereas the set V contains N unlabeled vertices. For any pair of vertices with one vertex from each set, there is a connecting edge of color i with probability p_i, $i = 0, \ldots, m$, with p_0 as the probability the two nodes are disconnected. Multiple edges between nodes are not allowed.

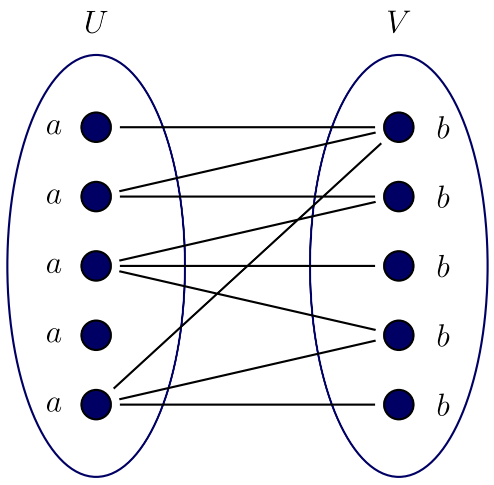
Definition 2. An *unlabeled bipartite graph* is a variation of a partially labeled bipartite graph where both sets U and V consist of unlabeled vertices.

In unlabeled bipartite graphs, for simplicity, in the sequel we assume there is only a single color for all nodes and that any two nodes from two different sets are connected with probability p.

To construct the K-layer neural network from the two substructures, one can think of it as made of a partially labeled bipartite graph for the first and last layers and a cascade of $K-2$ layers of unlabeled bipartite graphs



(a) Partially labeled bipartite graph with edge colors $\{0, 1, 2, 3, 4, 5\}$, where there is an edge of color 0 between a vertex from U and a vertex from V if they are not connected in the figure.



(b) Unlabeled bipartite graph.

Figure 2.1: Two structures for densely connected feedforward neural network layers.

for the remaining layers. An alternative construction is also possible: the first two layers are still a partially labeled bipartite graph 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.

2.2 Convolutional Neural Network

The convolutional neural network [27] is a very popular class of neural networks where each layer is connected to the next layer by a form of convolution of filters. This design of neural networks gained popularity because of its applicability in image, video, audio processing tasks. The key to the efficient design of CNN is because of its efficient use of parameters via parameter sharing, filters capturing local information in data, and equivariance to translation of input. Next, we present a simplified form of a convolutional layer, and then we show that it is equivariant to translations.

Let the feature maps in the input layer be represented by $f : \mathbb{Z} \mapsto \mathbb{R}$ and let $\psi : \mathbb{Z} \mapsto \mathbb{R}$ represent a filter. Then, the output of the neurons in the next layer is given by

$$[f \circledast \psi](x) = \sum_{y \in \mathbb{Z}} f(y)\psi(y - x) \tag{2.1}$$

Now we show the equivariance of a layer in CNN to translations. Let L_t denote translation of a layer by t. Then, from the definition of equivariance in Appendix A, we need to show that $[[L_t f] \circledast \psi](x) = L_t[[f \circledast \psi]](x)$.

$$\begin{aligned}
[[L_t f] \circledast \psi](x) &= \sum_{y \in \mathbb{Z}} f(y - t)\psi(y - x) \\
&= \sum_{y \in \mathbb{Z}} f(y)\psi(y + t - x) \\
&= \sum_{y \in \mathbb{Z}} f(y)\psi(y - (x - t)) \\
&= L_t[[f \circledast \psi]](x)
\end{aligned}$$

The property of equivariance in CNNs make the features translate in a predictable way when the input is translated which helps improve its performance compared to densely connected neural networks. The use of filters and convolution also reduces the number of free parameters available in a convolutional layer compared to a densely connected layer. Hence, the probabilistic model used in Sec. 2.1 cannot be directly used for this structure. A modified model that probabilistically captures the reduced number of free parameters in CNNs can improve on our algorithm for densely connected neural network layers. Next, we discuss a generalization of CNNs called GCNNs.

2.3 Group Equivariant Convolutional Neural Network

Group Equivariant Convolutional Neural Network is a generalization of CNNs in the sense that these networks are equivariant to group transformations instead of just translations. The basics about groups are provided in Appendix

A. As in CNNs, let the feature maps in the input layer be represented by $f : G \mapsto \mathbb{R}$ and let $\psi : G \mapsto \mathbb{R}$ represent a filter. Then, the output of the neurons in the next layer of a GCNN for a group G is given by

$$[f \circledast \psi](g) = \sum_{h \in G} f(h)\psi(g^{-1}h) \tag{2.2}$$

Now, we show that GCNNs are equivariant to group transformations corresponding to any group G. That is for any transformation L_u corresponding to an element $u \in G$, we need to show that $[[L_u f] \circledast \psi](g) = L_u[[f \circledast \psi]](g)$, where $[[L_u f] \circledast \psi](g) = \sum_{h \in G} f(u^{-1}h)\psi(g^{-1}h)$, and $L_u[[f \circledast \psi]](g) = \sum_{h \in G} f(h)\psi((u^{-1}h)^{-1}g)$. The proof for this follows:

$$\begin{aligned}
[[L_u f] \circledast \psi](g) &= \sum_{h \in G} f(u^{-1}h)\psi(g^{-1}h) \\
&= \sum_{h \in G} f(h)\psi(g^{-1}uh) \\
&= \sum_{h \in G} f(h)\psi((u^{-1}h)^{-1}g) \\
&= L_u[[f \circledast \psi]](g)
\end{aligned}$$

CHAPTER 3

REPRESENTING PARTIALLY LABELED BIPARTITE GRAPHS

We first compute the entropy bound for representing partially labeled bipartite graphs, then introduce a universal algorithm for approaching the bound, and finally an inference algorithm that need not fully decompress to operate.

3.1 Entropy Bound

Consider a matrix representing the edges in a partially labeled bipartite graph, such that each row represents an unlabeled node from V and each column represents a node from U. A non-zero matrix element i indicates there is an edge between the corresponding two nodes of color i, whereas a 0 indicates they are disconnected. Observe that if the order of the rows of this matrix is permuted (preserving the order of the columns), then the corresponding bipartite graph remains the same. That is, to represent the matrix, the *order of rows does not matter*. Hence the matrix can be viewed as a multiset of vectors, where each vector corresponds to a row of the matrix. Using these facts, we calculate the entropy of a partially labeled bipartite graph. To that end, we define the following terms.

Definition 3. Let $\mathcal{B}(N, p)$ be a random bipartite graph model in which graphs are randomly generated on two sets of vertices, U and V, having N labeled vertices each, with edges chosen independently between any two vertices belonging to different sets with probability p.

Definition 4. Let $\mathcal{B}_p(N, p)$ be a partially labeled random bipartite graph model generating graphs in the same way as a random bipartite graph model, except that the vertices in the set V in the generated graphs are unlabeled.

Definition 5. We say that a bipartite graph, b is *isomorphic* to a partially labeled bipartite graph b_p if b_p can be obtained by removing labels from all the vertices in set V of b, keeping all the edge connections the same. The set of all bipartite graphs, b, isomorphic to a partially labeled bipartite graph, b_p, is represented by $I(b_p)$.

Definition 6. The set of *automorphisms* of a graph, $Aut(b)$ for $b \in \mathcal{B}(N,p)$, is defined as an adjacency-preserving permutation of the vertices of a graph; $|Aut(b)|$ denotes the number of automorphisms of a graph b.

Definition 7. A graph g is called *asymmetric* if $|Aut(g)| = 1$; otherwise it is called *symmetric*.

Our proofs for entropy of random bipartite graphs follow that of [24] for entropy of random graphs.

Theorem 1. *For large N, and for all p satisfying $p \gg \frac{\ln N}{N}$ and $1-p \gg \frac{\ln N}{N}$, the entropy of a partially labeled bipartite graph, with each set containing N vertices and binary colored edges is $N^2 H(p) - \log_2(N!) + o(1)$, where $H(p) = p\log_2 \frac{1}{p} + (1-p)\log_2 \frac{1}{1-p}$, and the notation $a \gg b$ means $b = o(a)$.*

Proof. For a randomly generated bipartite graph, $b \in \mathcal{B}(N,p)$ with k edges, we have

$$P(b) = p^k (1-p)^{(N^2-k)}$$

Now, for each $b_p \in \mathcal{B}_p(N,p)$, there exist $|I(b_p)|$ corresponding $b \in \mathcal{B}(N,p)$ that are isomorphic to b_p. Hence,

$$P(b_p) = |I(b_p)| P(b)$$

Considering only the permutations of vertices in the set V, we have a total of $N!$ permutations. Given that each partially labeled graph b_p corresponds to $|I(b_p)|$ number of bipartite graphs, and each bipartite graph $b \in \mathcal{B}(N,p)$ corresponds to $|Aut(b)|$ (which is equal to $|Aut(b_p)|$) number of adjacency-preserving permutations of vertices in the graph, from [32, 33] one gets that:

$$N! = |Aut(b_p)| \times |I(b_p)|$$

By definition, the entropy of a random bipartite graph, $H_{\mathcal{B}}$, is $N^2 H(p)$ where $H(p) = p\log_2 \frac{1}{p} + (1-p)\log_2 \frac{1}{1-p}$. The entropy of a partially labeled graph is:

$$
\begin{aligned}
H_{\mathcal{B}_p} &= -\sum_{b_p \in \mathcal{B}_p(N,p)} P(b_p) \log_2 P(b_p) \\
&= -\sum_{b_p \in \mathcal{B}_p(N,p)} |I(b_p)| P(b) \log_2 (|I(b_p)| P(b)) \\
&= -\sum_{b \in \mathcal{B}(N,p)} P(b) \log_2 P(b) - \sum_{b_p \in \mathcal{B}_p(N,p)} P(b_p) \log_2 |I(b_p)| \\
&= -\sum_{b \in \mathcal{B}(N,p)} P(b) \log_2 P(b) - \sum_{b_p \in \mathcal{B}_p(N,p)} P(b_p) \log_2 \tfrac{N!}{|Aut(b_p)|} \\
&= H_{\mathcal{B}} - \log_2 N! + \sum_{b_p \in \mathcal{B}_p(N,p)} P(b_p) \log_2 |Aut(b_p)| \\
&= H_{\mathcal{B}} - \log_2 N! + \sum_{b_p \in \mathcal{B}_p(N,p) \text{ is symmetric}} P(b_p) \log_2 |Aut(b_p)| + \\
&\quad \sum_{b_p \in \mathcal{B}_p(N,p) \text{ is asymmetric}} P(b_p) \log_2 |Aut(b_p)|
\end{aligned}
$$

Now [34] shows that for all p satisfying the conditions in this theorem, a random graph $\mathcal{G}(N,p)$ on N vertices with edges occurring between any two vertices with probability p is symmetric with probability $O(N^{-w})$ for some positive constant w. We have stated and proved Lem. 17 in Appendix B to provide a similar result on symmetry of random bipartite graphs which will be used to compute its entropy.

Note that $|Aut(b_p)| = 1$ for asymmetric graphs, hence

$$\sum_{b_p \in \mathcal{B}_p(N,p) \text{ is asymmetric}} P(b_p) \log_2 |Aut(b_p)| = 0$$

We know that $N! = |Aut(b_p)| \times |I(b_p)|$, hence $|Aut(b_p)| \leq N!$. Therefore,

$$
\begin{aligned}
H_{\mathcal{B}_p} &\leq H_{\mathcal{B}} - \log_2 N! + \sum_{b_p \in \mathcal{B}_p(N,p) \text{ is symmetric}} P(b_p) N \log_2 N \\
&\leq H_{\mathcal{B}} - \log_2 N! + O(\tfrac{\log_2 N}{N^{w-1}})
\end{aligned}
$$

Hence, for any constant $w > 1$,

$$H_{\mathcal{B}_p} \leq N^2 H(p) - \log_2 N! + o(1)$$

This completes the proof. $\square$

We can also provide an alternate expression for the entropy of partially labeled graphs with m possible colors that will be amenable to comparison with the rate of a universal coding scheme.

Lemma 2. *The entropy of a partially labeled bipartite graph, with each set containing N nodes and edges colored with m possibilities is $N^2H(p) - \log_2(N!) + E[\sum_{i=1}^{(m+1)^N} \log_2(k_i!)]$, where $H(p) = \sum_{i=0}^{m} p_i \log_2 \frac{1}{p_i}$ and the k_is are non-negative integers that sum to N.*

Proof. As observed earlier, the adjacency matrix of a partially labeled bipartite graph is nothing but a multiset of vectors. From [21], we know that the empirical frequency of all elements of a multiset completely describes it. Each cell of the vector can be filled in $(m+1)$ ways corresponding to m colors or no connection (color 0), hence there can be in total $(m+1)^N$ possible vectors. The probability of a vector with the ith element having K_i appearances is:

$$\Pr[K_i = k_i] = \binom{N}{k_0, k_1, \ldots, k_{(m+1)^N}} \prod_{i=1}^{(m+1)^N} \pi_i^{k_i}$$

Here, π_i is the probability of occurrence of each of the possible vectors. In the ith vector, let the number of edges with color j be n_j. Then, $\pi_i = \prod_{j=0}^{m} p_j^{n_j}$. Hence, the entropy of the multiset is:

$$E[\log_2 \tfrac{1}{\Pr[K_i=k_i]}] = E\left[\sum \log_2 k_i!\right] + E\left[\sum k_i \log_2 \tfrac{1}{\pi_i}\right] - \log_2 N!$$

and

$$E[\sum k_i \log_2 \tfrac{1}{\pi_i}] = E\left[\sum_{(n_0,n_1,\ldots,n_m)} \left(n_{(n_0,n_1,\ldots,n_m)} \left(\sum_{j=0}^{m} n_j \log_2 \tfrac{1}{p_j}\right)\right)\right]$$

where $n_{(n_0,n_1,\ldots,n_m)}$ represents the number of vectors having n_j edges of color j. By linearity of expectation and rearranging terms, we get:

$$\sum_{(n_0,n_1,\ldots,n_m)} \sum_{j=0}^{m} \log_2 \tfrac{1}{p_j} E[n_j n_{(n_0,n_1,\ldots,n_m)}]$$

Now,

$$\Pr[n_{(n_0,n_1,\ldots,n_m)} = l] = \binom{N}{l}\left(\binom{N}{n_0,\ldots,n_m}\prod_{j=0}^{m} p_j^{n_j}\right)^{l}\left(1-\binom{N}{n_0,\ldots,n_m}\prod_{j=0}^{m} p_j^{n_j}\right)^{N-l}$$

$$\Rightarrow E[n_j n_{(n_0,n_1,\ldots,n_m)}] = n_j N\left(\binom{N}{n_0,n_1,\ldots,n_m}\prod_{j=0}^{m} p_j^{n_j}\right)$$

Thus,

$$\begin{aligned} E[\textstyle\sum k_i \log_2 \frac{1}{\pi_i}] &= \sum_{j=0}^{m} N \log_2 \tfrac{1}{p_j}\left(\sum_{(n_0,n_1,\ldots,n_m)} n_j\left(\binom{N}{n_0,n_1,\ldots,n_m}\prod_{j=0}^{m} p_j^{n_j}\right)\right) \\ &= \sum_{j=0}^{m} N^2 p_j \log_2 \tfrac{1}{p_j} = N^2 H(p) \end{aligned}$$

□

3.2 Universal Lossless Compression Algorithm

Next we present Alg. 1, a universal algorithm for compressing a partially labeled bipartite graph based on arithmetic coding, and its performance analysis.

Lemma 3. *If Alg. 1 takes L bits to represent the partially labeled bipartite graph, then $E[L] \leq N^2H(p) - \log_2 N! + E[\sum_{i=1}^{(m+1)^N} \log_2 k_i!] + 2$.*

Proof. We know, for any node encoded with α with the encodings of its child nodes $(\alpha_0, \alpha_1, \ldots, \alpha_m)$, that $(\alpha_0, \alpha_1, \ldots, \alpha_m)$ is distributed as a multinomial distribution, $\mathcal{M}(\alpha_0, \alpha_1, \ldots, \alpha_m; \alpha, P)$. So, using arithmetic coding to encode all the nodes, the expected number of bits required to encode all the nodes is

$$E\left[\sum \log_2 \frac{1}{\alpha! \prod_{i=0}^{m} \frac{(p_i)^{\alpha_i}}{\alpha_i!}}\right] \tag{3.1}$$

Here, the summation is over all non-zero nodes of the $(m+1)$-ary tree. Hence (3.1) can be simplified as

$$E[\textstyle\sum \alpha_i \log_2 \frac{1}{p_i}] + E[\sum \log_2 \alpha_i!] - \log_2 N!$$

Algorithm 1 Compressing a partially labeled bipartite graph.

```
Encode the total number of multisets in the root node of an $(m+1)$-ary tree using an integer code and initialize depth, $d = 1$.
Form $m + 1$ child nodes of the root node, and use arithmetic code to encode the $i$th child node with the number $x_i$, the number of vectors with $d$th cell having the $i$th color under the multinomial distribution. The vector $(x_{d,0}, x_{d,1}, \ldots, x_{d,m})$ follows a multinomial distribution $\mathcal{M}(x_{d,0}, x_{d,1}, \ldots, x_{d,m}; N, P)$, where $P$ represents the probability vector $(p_0, p_1, \ldots, p_m)$. Increase depth by 1.
while $d \leq N$ do
  for each of the nodes at the current depth do
    Form $m + 1$ child nodes of the current node (say, the current node is encoded with the number $\alpha$), and use arithmetic code to encode the child node of color $i$ with the number $\alpha_i$, where $\alpha_i$ represents the number of vectors with the $d$th column having color $i$ and all previous columns from 1 to $d$ having the same colors in the same order as that of the ancestor nodes of the child node starting from the root node. Here, $(\alpha_0, \alpha_1, \ldots, \alpha_m)$ follows a multinomial distribution $\mathcal{M}(\alpha_0, \alpha_1, \ldots, \alpha_m; \alpha, P)$.
  end for
  increase the depth by 1.
end while
```

When the term $E[\sum \log_2 \alpha_i!]$ is summed over all nodes, then all terms except those corresponding to the nodes of depth $N+1$ cancel, i.e. $E[\sum_{i=1}^{(m+1)^N} \log_2 (k_i!)]$. Similarly, the term $E[\sum \alpha_i \log_2 \frac{1}{p_i}]$ can be simplified as $N^2 \sum_{i=0}^{m} p_i \log_2 \frac{1}{p_i}$, since in the adjacency matrix of the graph, each cell can have colors from 0 to m with probability p_i, and for each color i, the expected number of cells having color i is $N^2 p_i$. Thus, we find

$$E\left[\sum \log_2 \frac{1}{\alpha! \prod_{i=0}^{m} \frac{(p_i)^{\alpha_i}}{\alpha_i!}}\right] = N^2 H(p) - \log_2 (N!) + E\left[\sum_{i=1}^{(m+1)^N} \log_2 k_i!\right]$$

Since we are using an arithmetic coder, it takes at most 2 extra bits [35, Ch. 13.3]. □

Theorem 4. *The expected compressed length generated by Alg. 1 is within 2 bits of the entropy bound.*

Proof. The result follows from Lem. 2 and Lem. 3 by comparing the entropy expression of a partially labeled random bipartite graph with the expected length in using Alg. 1. □

Theorem 4 states that space saving using this method can be made close to the theoretical limit. However, the theoretical limit in itself depends on the value of N, and hence analysis of the theoretical limit directly gives us the amount of space saving obtained. Note that the theoretical limit tells us that the space saving can be as much as $N \log N$ for large N for partially labeled bipartite graphs with each layer having N nodes, however, since the size of the graph is $O(N^2)$, the fraction of bits saved reduces as N increases. On the other hand, for small values of N, the theoretical limit does not allow us to save around $N \log N$ bits. Hence there is a trade-off between the amount of bits saved and the fraction of bits saved, i.e. for small values of N, the fraction of bits saved is more whereas as N increases, the fraction of bits saved decreases but the amount of bits saved increases.

3.3 Inference Algorithm

Algorithm 1 achieves near-optimal compression of partially labeled bipartite graphs, but we also wish to use such graphs as two-layered neural networks *without fully decompressing.* We next present Alg. 2 to directly use compressed graphs for the inference operations of two-layered neural networks. Structures that take space equal to the information-theoretic minimum with only a little bit of redundancy while also supporting various relevant operations on them are called *succinct structures* [3] as defined next.

Definition 8. If L is the information-theoretic minimum number of bits required to store some data, then we call a structure *succinct* if it represents the data in $L+o(L)$ bits, while allowing relevant operations on the compressed data.

Algorithm 2 Inference algorithm for compressed network.

```
1: **Input:** $X = [x_0, x_1, \ldots, x_{N-1}]$, the input vector to the neural network,
   and $\mathfrak{L}$, the compressed representation of the partially labeled bipartite
   graph obtained from Alg. 1.
2: **Output:** $Y = [y_0, y_1, \ldots, y_{N-1}]$, the output vector of the neural network,
   and $\mathfrak{L}$, the compressed representation as obtained from input.
3: **Initialize:** $Y = [y_0, y_1, \ldots, y_{N-1}] = [0, 0, \ldots, 0]$, $d = 0$, the number of
   neurons processed at the current depth, $j = 0$, an empty queue $Q$, and
   an empty string $\mathfrak{L}_1$ which would return the compressed representation
   $\mathfrak{L}$ once the algorithm has executed. Let $w_i$ represent the weight corre-
   sponding to color $i$.
4: Enqueue $Q$ with $N$, decoded from $\mathfrak{L}$ using integer coding.
5: **while** $Q$ is not empty and $d \leq N-1$ **do**
6:    $f = Q.pop()$.
7:    $i = 0$.
8:    **while** $i \leq m$ and $f > 0$ **do**
9:       Using arithmetic decoding, decode the child node of $f$ from $\mathfrak{L}$ cor-
         responding to color $i$ and store it as $c$.
10:      Encode $c$ back in $\mathfrak{L}_1$ using arithmetic coding.
11:      Enqueue $c$ in $Q$.
12:      Add $x_d \times w_i$ to each of $y_j$ to $y_{(j+c-1)}$.
13:      $j = (j + c) \bmod N$.
14:      **if** $j$ equals 0 and at least one non-zero node has been processed at
         the current depth **then**
15:         $d = d + 1$.
16:      **end if**
17:      $i = i + 1$.
18:   **end while**
19: **end while**
20: Update the $Y$ vector using the required activation function.
```

Algorithm 2 is a breadth-first search algorithm, which traverses through the compressed tree representation of the two-layered neural network and updates the output of the neural network, say Y, simultaneously. Note that the Y vector obtained from Alg. 2 is a permutation of the original $\tilde{Y}$ vector obtained from the original uncompressed network. Observe that each element of $\tilde{Y}$ has a corresponding vector indicating its connection with the input to the neural network, say X, and when all these elements are sorted in a decreasing manner based on these connections, it gives Y. This happens due to the design of Alg. 2 in giving the same Y vector independent of the arrangement in $\tilde{Y}$.[1]

Proposition 5. *Inference output Y obtained from Alg. 2 is a permutation of $\tilde{Y}$, the output from the uncompressed neural network representation.*

Proof. We need to show that the Y obtained from Alg. 2 is a permutation of $\tilde{Y}$, obtained by direct multiplication of the weight matrix with the input vector and passed through the activation function without any compression. 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. 2, 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. □

Proposition 6. *The additional dynamic space requirement of Alg. 2 is $O(N)$.*

Proof. It can be seen that Alg. 2 uses some space in addition to the compressed data. The symbols decoded from $\mathfrak{L}$ are encoded into $\mathfrak{L}_1$, hence, the combined space taken by both of them at any point in time remains almost the same as the space taken by $\mathfrak{L}$ at the beginning of the algorithm. However, the main dynamic space requirement is because of the decoding of individual nodes, and the queue, Q. Clearly, the space required for Q, storing up to two depths of nodes in the tree, is much more than the space required for decoding a single node.

[1] Based on this invariance in the output of the compressed neural network, we can rearrange the weights of the next layers of the neural network accordingly before compressing them to get a K-layered neural network with the desired output as done in Ch. 5.

We next show that 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 (with a small modification to be able to encode 0 as well) 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 values stored in the child nodes of non-zero tree nodes at some depth d of the tree, where $k \leq (m+1)N$. If $k < (m+1)N$, let $\alpha_{k+1}, ..., \alpha_{(m+1)N}$ be all zeros. 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 [36]. Thus, the arithmetic-geometric inequality implies

$$S \leq 2\left(\sum_{i=0}^{(m+1)N} 2\log_2(\alpha_i + 1) + 1\right) \leq 2N(m+1) + 4N(m+1)\log_2(\tfrac{m+2}{m+1}).$$

□

Theorem 7. *The compressed representation formed in Alg. 1 is succinct in nature.*

Proof. From Prop. 5 and Prop. 6 we know that the additional dynamic space required for Alg. 2 is $O(N)$, while the entropy of a partially labeled bipartite graph is $O(N^2)$. Thus, from the definition of succinctness, it follows that the structure is succinct. □

Next, we will find the time complexity of Alg. 2.

Proposition 8. *The time complexity of Alg. 2 is $O(mN^2)$.*

Proof. The time taken by Alg. 2 is the sum of time taken while decompressing the nodes and then compressing back each node of the tree, and computing the output using the decompressed node values. Assuming that multiplication takes constant time, the time taken for performing computations to get the output Y is $O(N^2)$, since for any $i \in \{0, \ldots, N-1\}$, x_i is multiplied to y_j for all $j \in \{0, \ldots, N-1\}$ at most once. The tasks of compression and decompression essentially take the same time, hence we will simply show that the time taken for compression is $O(mN^2)$. Encoding a tree node formed in Alg. 1 having value K with its parent node having value N, where $K \in \{0, \dots, N\}$, using arithmetic coding involves forming the cumulative distribution table of K in $O(N)$ time and finding the interval corresponding to K in the distribution table in $O(1)$ time. Hence, in the tree formed in Alg. 1, compressing a node having parent node with value N takes time $O(N)$ time. Now, there can be at most $m+1$ nodes with any particular parent node. Thus, compression of the tree using arithmetic coding will take $O((m+1)T)$ time, where T is the sum of all the node values in the tree. Also, note that the sum of node values in any layer can be at most N and the depth of the tree can at most be N, hence $T \leq N^2$. Thus, the time complexity of Alg. 2 is $O((m+1)N^2)$. $\square$

CHAPTER 4

UNLABELED BIPARTITE GRAPHS

Next we consider an unlabeled bipartite graph for which we construct the adjacency matrix similarly as before, but now the possible entries in each cell will be binary corresponding to whether or not there is an edge. We first compute the entropy bound for representing unlabeled bipartite graphs, and then introduce a universal algorithm for approaching the bound.

4.1 Entropy Bound

Although the structure is slightly different from the previous case, it also has some interesting properties. The connectivity pattern is independent of the order of the row vectors and column vectors in this bipartite adjacency matrix. We say that a matrix has undergone a row permutation if the order of the rows of the matrix is changed while keeping the order of cells in each row unchanged. Similarly, we say that a matrix has undergone a column permutation if the order of the columns of the matrix is changed while keeping the order of cells in each column unchanged. We say that a matrix has undergone a *valid* rearrangement is if it has undergone a sequence of row and column permutations. Note that under any valid rearrangement, the unlabeled bipartite graph remains unchanged. Let A represent the adjacency matrix of a bipartite graph and a_{ij} be the cell in the matrix at row i and column j. Say a valid rearrangement of A transforms it to some matrix, A', then, if a cell at row i and column j of A has moved to row k and column l of the matrix A' after transformation, then note that the set of cells in row i of A is the same as the set of cells in row k of A'. We call this set of cells at row i the row block corresponding to the cell a_{ij}, since this set of cells corresponding to a_{ij} does not change under any valid rearrangement. Similarly, we call the set of cells at column j, the column block corresponding to the cell a_{ij}.

We will next show that the entropy of an unlabeled random bipartite graph is $N^2H(p) - 2\log_2(N!) + o(1)$. To that end, we need the following definitions.

Definition 9. Let $\mathcal{B}_u(N, p)$ be an unlabeled random bipartite graph model generating graphs in the same way as a random bipartite graph model, except that the vertices in both the sets, U and V, are unlabeled, but the sets U and V themselves remain labeled, i.e. two sets of unlabeled vertices having the same edge connections as that of a random bipartite graph.

Definition 10. We say b is *isomorphic* to b_u if b_u can be formed by removing labels from all the vertices of b, keeping all the edge connections the same. The set of all bipartite graphs isomorphic to an unlabeled bipartite graph, b_u, is represented by $I(b_u)$.

Theorem 9. *For large N, and for all p satisfying $p \gg \frac{\ln n}{n}$ and $1 - p \gg \frac{\ln n}{n}$, the entropy of an unlabeled bipartite graph, with each set containing N vertices and binary colored edges is $N^2H(p) - 2\log_2(N!) + o(1)$, where $H(p) = p\log_2\frac{1}{p} + (1-p)\log_2\frac{1}{1-p}$, and the notation $a \gg b$ means $b = o\,(a)$.*

Proof. From Thm. 1, we know that for a graph $b \in \mathcal{B}(N, p)$ with k edges,

$$P(b) = p^k(1-p)^{(N^2-k)}$$

For each $b_u \in \mathcal{B}_u(N, p)$, there exist $|I(b_u)|$ number of corresponding $b \in \mathcal{B}(N, p)$. Thus we have

$$P(b_u) = |I(b_u)|P(b)$$

Considering the permutations of vertices in the sets V and U themselves, we have a total of $(N!)^2$ permutations. Given that each unlabeled graph b_u corresponds to $|I(b_u)|$ number of bipartite graphs, and each bipartite graph $b \in \mathcal{B}(N, p)$ corresponds to $|Aut(b)|$ (which is equal to $|Aut(b_u)|$), we get the number of adjacency-preserving permutations of vertices in the graph, from [32, 33], as:

$$(N!)^2 = |Aut(b_u)| \times |I(b_u)|$$

We also know that the entropy of random bipartite graph, $H_{\mathcal{B}}$, is $N^2H(p)$.

The entropy of an unlabeled graph is:

$$\begin{aligned}
H_{\mathcal{B}_u} &= -\sum_{b_u \in \mathcal{B}_u(N,p)} P(b_u) \log_2 P(b_u) \\
&= -\sum_{b_u \in \mathcal{B}_u(N,p)} |I(b_u)| P(b) \log_2 \left(|I(b_u)| P(b)\right) \\
&= -\sum_{b \in \mathcal{B}(N,p)} P(b) \log_2 P(b) - \sum_{b_u \in \mathcal{B}_u(N,p)} P(b_u) \log_2 |I(b_u)| \\
&= -\sum_{b \in \mathcal{B}(N,p)} P(b) \log_2 P(b) - \sum_{b_u \in \mathcal{B}_u(N,p)} P(b_u) \log_2 \tfrac{(N!)^2}{|Aut(b_u)|} \\
&= H_{\mathcal{B}} - 2\log_2 N! + \sum_{b_u \in \mathcal{B}_u(N,p)} P(b_u) \log_2 |Aut(b_u)| \\
&= H_{\mathcal{B}} - 2\log_2 N! + \sum_{b_u \in \mathcal{B}_u(N,p) \text{ is symmetric}} P(b_u) \log_2 |Aut(b_u)| + \\
&\quad \sum_{b_u \in \mathcal{B}_u(N,p) \text{ is asymmetric}} P(b_u) \log_2 |Aut(b_u)|
\end{aligned}$$

We will next use a result, Lem. 18 in Appendix B, on symmetry of random bipartite graphs to compute entropy.

Note that $|Aut(b_u)| = 1$ for asymmetric graphs and so:

$$\sum_{b_u \in \mathcal{B}_u(N,p) \text{ is asymmetric}} P(b_u) \log_2 |Aut(b_u)| = 0$$

We know that $(N!)^2 = |Aut(b_u)| \times |I(b_u)|$, hence $|Aut(b_u)| \leq (N!)^2$. Therefore,

$$\begin{aligned}
H_{\mathcal{B}_u} &\leq H_{\mathcal{B}} - 2\log_2 N! + \sum_{b_u \in \mathcal{B}_u(N,p) \text{ is symmetric}} P(b_u) 2N \log_2 N \\
&\leq H_{\mathcal{B}} - 2\log_2 N! + O\left(\tfrac{\log_2 N}{N^{w-1}}\right)
\end{aligned}$$

Further, note that $H_{\mathcal{B}} = N^2 H(p)$ where $H(p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p}$. Hence, for any constant $w > 1$,

$$H_{\mathcal{B}_u} \leq N^2 H(p) - 2\log_2 N! + o(1)$$

□

4.2 Universal Lossless Compression Algorithm

In this section, we provide a lossless compression algorithm for unlabeled bipartite graph which is optimal up to the second-order term. Algorithm 3 takes the adjacency matrix of an unlabeled bipartite graph as input and outputs two tree structures which are invariant to any valid rearrangement of the graph. Then these trees are compressed as follows: we perform a breadth first search on each of the trees and the child nodes of a node with value, say N_x, are first stored using $\lceil \log_2 (Nx + 1) \rceil$ bits and then the bit-stream produced after the completion of the breadth first search is compressed using an arithmetic encoder. Note that binomial distribution has been used for arithmetic coding, with p as the probability of existence of an edge between any two nodes of the bipartite graph and $q = 1 - p$ as the probability that the two nodes are disconnected.

It can be observed that the structure of the trees formed in Alg. 3 is the same as in [24] except that there are two trees in our algorithm and the first tree does not lose an element from the root node on its first division. Let us now define a tree structure which will be useful for the analysis of the performance of the algorithm.

Definition 11. Let $\mathcal{T}_{n,d,p}$ be a class of random binary trees such that any tree $T_{n,d,p} \in \mathcal{T}_{n,d,p}$ has depth $(n-1)$ and is generated in the following way: (1) The root node is assigned the value n and placed at depth 0. (2) If $d > 0$, then starting from depth, $t = 0$ to $t = d - 1$, divide each of the nodes with non-zero values at the current depth into two child nodes such that the sum of the values assigned to the child nodes is equal to that of the parent node (say N), and the left child node has value N_1 distributed as binomial distribution, $N_1 \sim Binomial(N, p)$. Else, if $d = 0$, skip this step. (3) Starting from depth $t = d$ to $t = n - 2$, subtract the value of the leftmost node with non-zero value and divide each of the non-zero nodes at the current depth into two child nodes in the same way as in the previous step using the updated node values after subtraction. That is, the sum of the values of the child nodes is equal to that of the updated value of the parent node, and the left child node has value assigned to it using binomial distribution. We write $\mathcal{T}_{n,d,p}$ as $\mathcal{T}_{n,d}$ when p is clear from context, and we use the notations $T_{n,0}$ and T_n interchangeably.

Algorithm 3 Compressing an unlabeled bipartite graph.

```
1: Choose any cell containing 1 (call it 1-cell) from the adjacency matrix (or any cell containing 0 (0-cell) only if no 1-cell is available) and using valid rearrangements make this cell the top left element of the matrix. Call it the parent cell. Initially, all cells are unmarked.
2: Form two trees $t_1$ and $t_2$, and store $N$ in the root nodes of each of the trees. Initialize depth, $d = 1$.
3: while depth of $t_1 \leq N + 1$ do
4:     Divide every non-empty leaf node at the current depth of tree $t_1$ into two child nodes. The left child denotes the number of 1-cells that are unmarked in the column block containing the parent cell; similarly the right child denotes the remaining 0-cells that are unmarked.
5:     Mark all unmarked cells in the column block containing the parent cell.
6:     Remove an element from the leftmost node of the tree $t_2$.
7:     Choose any cell from the newly formed leftmost child of the tree $t_1$ as the parent cell.
8:     Divide all the leaf nodes at the current depth of the tree $t_2$ into two child nodes. The left child denotes the number of unmarked 1-cells in the row block containing the parent cell; similarly the right child denotes the remaining 0-cells that are unmarked.
9:     Choose any cell from the newly formed leftmost child of the tree $t_2$ as the parent cell.
10:    Mark all the unmarked cells in the row block containing the parent cell.
11:    Remove an element from the leftmost node of the tree $t_1$.
12:    Increase depth of $t_1$ and $t_2$ by 1.
13: end while
```

Let N_x be the number of elements in some node x of either of the trees formed in Alg. 3 (say T, where T can be t_1 or t_2 formed in the algorithm). Then the total number of bits required for encoding the tree before using arithmetic coding is $\sum_{x\in T \text{ and } N_x\geq 1}\lceil \log_2(N_x+1)\rceil$. Define $L_1 = \sum_{x\in T \text{ and } N_x>1}\lceil \log_2(N_x+1)\rceil$ and $L_2 = \sum_{x\in T \text{ and } N_x=1}\lceil \log_2(N_x+1)\rceil$. Let $\hat{L_1}$ and $\hat{L_2}$ be the length of bit-streams corresponding to L_1 and L_2 respectively after being compressed using arithmetic coding. So, the total expected bit length is $E[L_1]+E[L_2]$ before using arithmetic coding, and $E\left[\hat{L_1}\right]+E\left[\hat{L_2}\right]$ after using arithmetic coding. Now define

$$a_{n,d} = E\left[\sum_{x\in T_{n,d} \text{ and } N_x>1}\lceil \log_2(N_x+1)\rceil\right]$$

$$b_{n,d} = \sum_{x\in T_{n,d}} N_x - \sum_{x\in T_{n,d} \text{ and } N_x=1} N_x$$

Now we bound the compression performance of Alg. 3. The proof for this bound is based on a theorem for compression of graphical structures [24] and before stating our result and its proof, we recall two lemmas from there.

Lemma 10. *For all integers $n \geq 0$ and $d \geq 0$,*

$$a_{n,d} \leq x_n$$

where x_n satisfies $x_0 = x_1 = 0$ and for $n \geq 2$

$$x_n = \lceil \log_2(n+1)\rceil + \sum_{k=0}^{n}\binom{n}{k}p^k q^{n-k}(x_k + x_{n-k})$$

Lemma 11. *For all $n \geq 0$ and $d \geq 0$,*

$$b_{n,d} \geq y_n - \frac{n}{2},$$

such that y_n satisfies $y_0 = 0$ and for $n \geq 0$,

$$y_{n+1} = n + \sum_{k=0}^{n}\binom{n}{k}p^k q^{n-k}(y_k + y_{n-k})$$

Theorem 12. *If an unlabeled bipartite graph can be represented by Alg. 3 in L bits, then $E[L] \leq N^2 H(p) - 2N\log_2(N) + 2(c + \Phi(\log_2(N+1)))(N+1) + o(N)$, where c is an explicitly computable constant, and $\Phi(\log_2(N+1))$ is a fluctuating function with a small amplitude independent of N.*

Proof. We need to find the expected value of the sum of all the encoding-lengths in all nodes of both trees. The expected value of length of encoding for both trees can be upper-bounded by an expression provided in [24].

Let us formally prove that both encodings are upper-bounded by this expression. If $E[L_{t_1}]$ and $E[L_{t_2}]$ are the number of bits required to represent trees t_1 and t_2, respectively, then the following equations hold.

$$\begin{aligned} E[L_{t_1}] &= a_{N,1} + \tfrac{N(N+1)}{2} - b_{N,1} \\ E[L_{t_2}] &= a_{N,0} + \tfrac{N(N-1)}{2} - b_{N,0} \end{aligned}$$

Similarly, $E\left[\hat{L_{t_1}}\right]$ and $E\left[\hat{L_{t_2}}\right]$ are the number of bits required to represent trees t_1 and t_2 after using arithmetic coding, respectively. Using Lem. 10 and Lem. 11, and bounds on x_n and y_n from [24] it follows that for any $d \geq 0$:

$$\begin{aligned} E\left[\hat{L_{t_1}}\right] &\leq \tfrac{N(N+1)}{2} H(p) - N\log_2 N + (c + \Phi(\log_2(N+1)))(N+1) + o(N) \\ E\left[\hat{L_{t_2}}\right] &\leq \tfrac{N(N-1)}{2} H(p) - N\log_2 N + (c + \Phi(\log_2(N+1)))(N+1) + o(N) \end{aligned}$$

Hence, the sum:

$$E\left[\hat{L}_{t_1}\right] + E\left[\hat{L}_{t_2}\right] \leq N^2 H(p) - 2N\log_2 N + 2(c + \Phi(\log_2(N+1)))(N+1) + o(N)$$

where c is an explicitly computable constant and $\Phi(\log(N+1))$ is a fluctuating function with a small amplitude independent of N. This completes the proof. □

It can be observed that by using Alg. 3 for unlabeled bipartite graphs, we save roughly $N\log_2 N$ bits when compared to compressing partially labeled bipartite graph using Alg. 1.

CHAPTER 5

DEEP NEURAL NETWORKS

Now we return to the K-layer neural network model from Ch. 2. First we extend the algorithm for unlabeled bipartite graph to compress K-layered unlabeled graph, and then store the permutation of the first and last layers. This gives us an efficient compression algorithm for a K-layered neural network, saving around $(K-2) \times N \log_2 N$ bits compared to standard arithmetic coding of weight matrices. Algorithm 4 takes the feedforward neural network in the form of its weight matrices as input and outputs K tree structures which are invariant to any valid rearrangement of the weight matrices. Then these trees are compressed similar to unlabeled bipartite graphs in Ch. 3 as follows: we perform a breadth first search on each of the trees and the child nodes of a node with value, say N_x, are first stored using $\lceil \log_2 (Nx+1) \rceil$ bits and then the bit-stream produced after the completion of the breadth first search is compressed using an arithmetic encoder. The binomial distribution has been used for arithmetic coding, with p as the probability of existence of an edge between any two nodes of the bipartite graph and $q = 1 - p$ as the probability that the two nodes are disconnected.

5.1 Universal Lossless Compression Algorithm Using Unlabeled Bipartite Graphs

Theorem 13. *Let L be the number of bits required to represent a K-layer neural network model using Alg. 4. Then $E[L] \leq (K-1)N^2H(p) + (K-2)NH(p) - (K-2)N\log N + K(c + \Phi(\log(N+1)))(N+1) + o(N)$, where c is an explicitly computable constant, and $\Phi(\log(N+1))$ is a fluctuating function with a small amplitude independent of N.*

Proof. The encoding of Alg. 4 is similar to the encoding of Alg. 3. For all trees, the child nodes of any node with non-zero value N_x are stored using

Algorithm 4 Compressing a K-layer unlabeled graph.

```
 1: Form root nodes of $K$ binary trees $t_1, t_2, \ldots, t_K$ corresponding to $K$ layers
    of the neural network, and store $N$ in the root node of all the trees,
    corresponding to the $N$ neural network nodes in each of the layers.
 2: Initialize iteration number, $i = 1$, and layer number, $j = 1$. Let $\Gamma(j)$
    represent the set of indices of trees corresponding to layers neighboring
    to the $j$th layer of the neural network.
 3: while depth of $i \leq N$ do
 4:   while depth of $j \leq K$ do
 5:     Selection: Select a node of the neural network from layer $j$ that
        corresponds to one of the neural network nodes in the leftmost non-
        zero node of $t_j$ and subtract 1 from the leftmost non-zero node of
        $t_j$.
 6:     Division: Divide every non-empty leaf node of the trees $t_k$ for $k \in$
        $\Gamma(j)$ into two child nodes based on the connections of the neural
        network nodes corresponding to the leaf nodes with the selected
        node in the previous step. The left child denotes the number of
        neural network nodes not connected to the selected node; similarly
        the right child denotes the neural network nodes connected to the
        selected node.
 7:     Increment $j$ by 1.
 8:   end while
 9:   Increment $i$ by 1.
10: end while
```

$[\log_2 N_x + 1]$ bits. Let the number of bits required to encode the jth layer be L_j. These bits are further compressed using an arithmetic coder, which gives us, say, $\hat{L}_j$ bits for the jth layer. Observe that the trees for the first and Kth layer belong to $\mathcal{T}_{N,0}$ and $\mathcal{T}_{N,1}$ respectively. Hence, based on results from previous sections,

$$E\left[\hat{L}_1\right]+E\left[\hat{L}_K\right] \leq N^2 H(p) - 2N\log_2 N + 2(c+\Phi(\log_2{(N+1)}))(N+1)+o(N)$$

But the binary trees formed for the layers 2 to $K-1$ are different. Instead of a subtraction from the leftmost non-zero node at each division after the first d divisions as in a $\mathcal{T}_{n,d}$ type of tree, in these type of trees, let us call them $\mathcal{T}^2_{n,d}$ type of trees, subtraction takes place in every alternate division after the first d divisions. We will follow the same procedure for compression of t_2 to t_{K-1} as for t_1 and t_K, i.e. we will encode the child nodes of a node with value N_x with $[\log_2 N_x + 1]$ bits followed by an arithmetic coder. Now define,

$$\begin{aligned} a^2_{n,d} &= E\left[\sum_{x \in T^2_{n,d} \text{ and } N_x > 1} \lceil \log_2{(N_x+1)} \rceil\right] \\ b^2_{n,d} &= \sum_{x \in T^2_{n,d}} N_x - \sum_{x \in T^2_{n,d} \text{ and } N_x = 1} N_x \end{aligned}$$

We show that $a^2_{n,d} \leq x_n$ and $b^2_{n,d} \geq y_n - \frac{n}{2}$ for x_n and y_n as defined in Lem. 10 and Lem. 11, respectively. These are stated and proved as Lem. 19 and Lem. 20 in Appendix B.

Returning to the proof, since the trees t_i for $i \in \{2, \ldots, K-1\}$, are all of the same type, we will have the same expected length of coding for each of them. Let the expected encoding length for a tree t_i for $i \in \{2, \ldots, K-1\}$ before using arithmetic coding be $E\left[L_i\right]$, and that after using arithmetic coding be $E\left[\hat{L}_i\right]$. Then,

$$E\left[L_i\right] = N(N+1) + a^2_{N,1} - b^2_{N,1}$$

Using upper bounds proved in Lem. 19 and Lem. 20, from [24], we know that

$$E\left[\hat{L}_i\right] \leq (N^2+N)H(p) - N\log_2 N + (c + \Phi(\log_2{(N+1)}))(N+1) + o(N)$$

where c is an explicitly computable constant and $\Phi(\log{(N+1)})$ is a fluctuating function with a small amplitude independent of N. Further, since we need to store the permutation of the input and output layers, we need to store another $2\lceil N\log_2 N\rceil$ bits. This completes the proof. $\square$

5.2 Universal Lossless Compression Algorithm Using Partially Labeled Bipartite Graphs

Now consider an alternative method to compress a deep neural network, using Alg. 1 iteratively to achieve efficient compression.

Theorem 14. *Let L be the number of bits required to represent a K-layer neural network model through iterative use of Alg. 1. Then $E[L] \leq (k-1)(N^2H(p) - \log(N!) + E[\sum_{i=1}^{(m+1)^N}\log{(k_i!)}]) + \log_2 N! + c$, where $H(p) = \sum_{i=0}^{m} p_i \log\frac{1}{p_i}$, the k_is are as defined in Lem. 2, and c is a constant representing the amount of additional bits required by an arithmetic coder for initiating and finishing encoding.*

Proof. If we focus only on the first two layers of the neural network model, then by Lem. 2, it can be compressed in less than $N^2H(p) - \log N! + E[\sum_{i=1}^{(m+1)^N}\log{(k_i!)}]$ number of bits. Once the first two layers are encoded, one can label the nodes of the second layer based on the relationship of its connectivity with the nodes of the first layer, and treat the second layer as a labeled layer. Also, the third layer is unlabeled and hence Alg. 1 can be used again to compress the second and third layer using less than $N^2H(p) - \log N! + E[\sum_{i=1}^{(m+1)^N}\log{(k_i!)}$ number of bits. This, can be repeated until all layers are encoded. Further, we also need to store the permutation of the outer layer of the neural network, which takes an additional $\log_2 N!$ bits.

Hence, iteratively encoding the K layers gives:

$$E[L_K] \leq (K-1)\left(N^2H(p) - \log N! + E[\sum_{i=1}^{(m+1)^N}\log{(k_i!)}] + \log_2 N! + c\right)$$

where c is the additional number of bits that an arithmetic coder takes to start and finish encoding. □

We have developed two different compression algorithms for feedforward neural networks. The compression algorithm based on partially labeled graph appears to be inefficient compared to the one based on unlabeled bipartite graph since after removing invariances from each layer, it treats the hidden layer as a labeled layer for compressing the next hidden layer, introducing some redundancy. However, both algorithms are asymptotically optimal up to the second-order term. Further, the algorithm based on the partially labeled graph is easier to implement and also enables easy updates in the compressed structure. Hence, in the next section, we provide an inference algorithm that makes use of compressed representation of a feedforward neural network generated using the iterative algorithm introduced in this section.

5.3 Inference Algorithm

Inference for a K-layered neural network is just an extension of Alg. 2. In particular, the output of Alg. 2 becomes the input for the next layers. However, one important point to consider in compression, so as to ensure the inference algorithm of the K-layered neural network still works, is to appropriately rearrange the weight matrices. Note that Alg. 2 outputs the Y in a specific pattern, i.e. the output Y is sorted based on the connections of output nodes with the input nodes; thus for the algorithm to work, we need to sort the weight matrix corresponding to the next layer accordingly before compressing them. Also, note that the last weight matrix connecting to the output layer of the K-layered neural network need not be compressed since it is desirable to preserve the ordering of the output layer nodes.

Theorem 15. *The compressed structure obtained by the iterative use of Alg. 1 is succinct.*

Proof. Since each layer is computed one at a time in inference and the extra space required during the inference task of a two-layered neural network is stored only temporarily, the extra dynamic space requirement for a K-layered remains the same as for the two-layered neural network described in Alg. 2.

Hence, the compressed representation for the K-layered neural network is succinct. □

Next we provide the time complexity for inference using Alg. 2 iteratively and compare it with inference on an uncompressed neural network.

Proposition 16. *The time complexity of Alg. 2 used iteratively on a K-layered neural network for inference is $O(mKN^2)$. The time complexity for inference on an uncompressed neural network is $O(KN^2)$.*

Proof. From Prop. 8, we already know that the time complexity of Alg. 2 is $O(mN^2)$. Clearly, iteratively using Alg. 2 K times takes $O(mKN^2)$ time. Further, each layer of an uncompressed neural network requires $O(N^2)$ computation due to matrix multiplication of a vector of size $1 \times N$ with a weight matrix of size $N \times N$. Hence, K such layers take $O(KN^2)$ time. □

CHAPTER 6

EXPERIMENTS

To validate and assess our neural network compression scheme, we trained feedforward neural networks using stochastic gradient descent on three datasets, and quantized them using different quantization schemes before using our lossless compression scheme. The three datasets used are the MNIST dataset [37], IMDB movie reviews sentiment classification dataset [38], and the Reuters-21578 dataset [39]. The weights of each of the trained networks were uniformly quantized using 17, 33, and 65 quantization levels in the interval $[-0.16, 0.16]$. 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 to get an accuracy of 95.9% on the test data. The test accuracy of the quantized networks are 87.1%, 94.3%, and 94.9% for quantization levels of 17, 33, and 65 respectively. Similarly, for the IMDB dataset, a feedforward neural network of dimension $1000 \times 128 \times 64 \times 2$ was trained which gives a test accuracy of 85.9%. The quantized networks give test accuracy of 77.9%, 84.7%, and 85.5% for quantization levels of 17, 33, and 65 respectively. For the Reuters-21578 dataset, we trained a feedforward neural network of dimension $1000 \times 200 \times 100 \times 46$ to get a test accuracy of 77.0%. The quantized networks give test accuracy of 72.9%, 75.9%, and 76.4% for quantization levels of 17, 33, and 65 respectively.

The weight matrices from the second-to-last layer were rearranged based on the weight matrices corresponding to the previous layers as needed for Alg. 2 to work. These matrices, except the last matrix connected to the output, were compressed using Alg. 1 to get the compressed network, and arithmetic coding was implemented by modification of an existing implementation.[1] The compressed network performed exactly as the original quantized network (as

[1]Nayuki, "Reference arithmetic coding," https://github.com/nayuki/Reference-arithmetic-coding, Nov. 2017. Our implementations can be found at `https://github.com/basusourya/DNN`

it should have) since our compression is lossless. We observe that the extra memory required for inference is negligible when compared to the size of the compressed network. Detailed results from the experiments and dynamic space requirements are described in Tab. 6.1, Tab. 6.2, and Tab. 6.3 for the MNIST, IMDB, and Reuters datasets respectively, where $H(p)$ is the empirical entropy calculated from the weight matrices.

In these tables, the term $MNH(p) - N \log_2 N$ represents an approximation to the theoretical bounds in Thm. 13 and 14 since computing the exact bounds is difficult. The parameters "Avg. queue length" and "Max. queue length" represent the average and maximum dynamic space requirements for Alg. 2 respectively. The fact that these two parameters have small values compared to the size of the network implies that inference without full decompression of the network takes marginal additional dynamic space.

Tables 6.4 and 6.5 measure the time needed for Alg. 2. Table 6.4 gives a comparison between time taken for inference using compressed and uncompressed neural networks. The experiments were run using a naive Python implementation on a system with 12GB RAM, Intel(R) Xeon(R) CPU @ 2.20GHz processor. Note that in Tab. 6.4 and 6.5, the neural networks are named after the data they were trained on and their quantization levels for conciseness, and that the number of parameters is the number of weights in a network. Table 6.5 provides the distribution of time taken by different components of Alg. 2. In particular, in Tab. 6.5 "% pmf computation" and "% arithmetic decoding + re-encoding" denote the percentage of time taken for computation of the pmf for arithmetic coder, and for decoding and re-encoding respectively. Results show that time taken for making inference using compressed networks is considerably higher than corresponding uncompressed neural networks, but seemingly not impractical on an absolute scale. We further investigate the time taken by different components of Alg. 2 in Tab. 6.5. It can be observed that roughly 90% of the time taken in Alg. 2 is due to arithmetic encoding/decoding and probability matrix computation. Arithmetic coding is an essential component of our inference algorithm 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 [40, 41]. Such efficient implementations would likely improve time required for our algorithms considerably.

Table 6.1: Experiments for the MNIST dataset for Alg. 1 and Alg. 2.

Shape of weight matrix ($M \times N$)	Quantization level	$MNH(p) - N\log_2 N$	Observed length (bits)	Avg. queue length (bits)	Max. queue length (bits)
$M = 784, N = 50$	17	152426	149994	150	257
	33	188286	188165	151	397
	65	223936	225998	155	778
$M = 50, N = 50$	17	9456	10254	152	255
	33	11743	11853	154	251
	65	14017	13480	180	396
$M = 50, N = 50$	17	9456	10304	156	290
	33	11743	11892	165	336
	65	14017	13465	194	569
$M = 50, N = 50$	17	9456	10383	153	245
	33	11743	12004	173	475
	65	14017	13688	178	520

Table 6.2: Experiments for the IMDB dataset Alg. 1 and Alg. 2.

Shape of weight matrix ($M \times N$)	Quantization level	$MNH(p) - N\log_2 N$	Observed length (bits)	Avg. queue length (bits)	Max. queue length (bits)
$M = 1000, N = 128$	17	436597	422241	384	625
	33	562773	548951	385	825
	65	689138	676129	389	1379
$M = 128, N = 64$	17	27615	41878	193	331
	33	35690	49486	204	618
	65	43778	56822	226	910

Table 6.3: Experiments for the Reuters dataset Alg. 1 and Alg. 2.

Shape of weight matrix ($M \times N$)	Quantization level	$MNH(p) - N\log_2 N$	Observed length (bits)	Avg. queue length (bits)	Max. queue length (bits)
$M = 1000, N = 200$	17	731756	711156	600	954
	33	927898	909230	602	1444
	65	1124189	1107635	606	1739
$M = 200, N = 100$	17	72664	87618	301	462
	33	92278	106227	307	822
	65	111907	124906	336	1481

Table 6.4: Comparison of inference time for compressed and uncompressed neural networks.

Network name	No. of parameters	Uncompressed inference time	Compressed inference time
MNIST17	46700	0.06 sec	2.30 sec
MNIST33	46700	0.06 sec	2.81 sec
MNIST65	46700	0.06 sec	3.34 sec
IMDB17	136192	0.17 sec	6.4 sec
IMDB33	136192	0.17 sec	7.41 sec
IMDB65	136192	0.17 sec	8.91 sec
Reuters17	220000	0.26 sec	10.14 sec
Reuters33	220000	0.26 sec	12.07 sec
Reuters65	220000	0.27 sec	14.99 sec

Table 6.5: Percentage time taken by different components of Alg. 2.

Network name	No. of parameters	% pmf computation	% arithmetic decoding + re-encoding
MNIST17	46700	12	82
MNIST33	46700	15	80
MNIST65	46700	19	76
IMDB17	136192	9	84
IMDB33	136192	11	83
IMDB65	136192	14	80
Reuters17	220000	10	83
Reuters33	220000	12	82
Reuters65	220000	16	79

CHAPTER 7

CONCLUSION

Data and models that are stored in memory and used for computation are often no longer of conventional type such as sequential texts or images, but rather could include structural data such as artificial neural networks, connectomes, phylogenetic trees, or social networks [24, 42]. Moreover there is growing interest in using neural network models for on-device intelligence and for scaling cloud-based intelligence, but high-performing deep neural networks are too large in size. To ameliorate this storage bottleneck, we have developed lossless compression algorithms for feedforward deep neural networks that make use of their particular structural invariances in inference and can act as a final stage for other lossy techniques [15]. Given that there may be limited prior knowledge on the statistics of synaptic weight and structure, our compression schemes are universal and yet asymptotically achieve novel entropy bounds. Further, we show that the proposed compressed representations are succinct and can be used for inference without complete decompression. These compression algorithms can also be directly used in fully connected layers of other variants of neural networks, such as convolutional neural networks or recurrent neural networks.

In future work, we plan to investigate optimal quantization of real-valued synaptic weights using ideas from functional quantization [43], but taking into account our novel form of entropy coding.

APPENDIX A

BASICS OF GROUP THEORY

First we will define a group and then define the notion of equivariance.

Definition 12. A group $(G, \circ)$ is a set G equipped with a binary operator $\circ$ such that the following axioms are satisfied for $(G, \circ)$.

- Closure: For any $g_1, g_2 \in G$, then $g_1 \circ g_2 \in G$.
- Associativity: For any $g_1, g_2, g_3 \in G$, then $g_1 \circ (g_2 \circ g_3) = (g_1 \circ g_2) \circ g_3$.
- Identity: There exists an $e \in G$, such that $g \circ e = e \circ g$ for all $g \in G$.
- Inverse: For any $g \in G$, there is a corresponding g^{-1}, such that $g \circ g^{-1} = g^{-1} \circ g = e$.

Next we define the notion of equivariance of a function ϕ with respect to a group G.

Definition 13. A function $\phi(\cdot) : \mathcal{X} \mapsto \mathcal{Y}$ is equivariant to a group G if $g \circ \phi(x) = \phi(g \circ x)$ for all $g \in G$ and $x \in \mathcal{X}$.

APPENDIX B

PROOFS

Lemma 17. *For all p satisfying $p \gg \frac{\ln N}{N}$ and $1 - p \gg \frac{\ln N}{N}$, a random partially bipartite graph is symmetric with probability $O(N^{-w})$ for any positive constant w.*

Proof. Define $B = (\{U, V\}, E)$, a partially labeled bipartite graph with two sets of vertices U and V and set of edges E. Let $\pi : U \cup V \to U \cup V$ be the permutation of vertices in the sets U and V. Further, since the vertices in U are labeled, we take $\pi(u) = u$ for $u \in U$. Following the definitions of [34], for a vertex $v \in V$, we define a defect of v with respect to π to be

$$D_\pi(v) = |\Gamma(\pi(v))\Delta\pi(\Gamma(v))|$$

where $\Gamma(v)$ is the set of neighbors of v and Δ denotes the symmetric difference of two sets, i.e., $A\Delta B = (A - B) \cup (B - A)$ for two sets A and B. Similarly, one can define a defect of B with respect to π to be

$$D_\pi(B) = \max_v D_\pi(v)$$

and the defect of a graph B can be defined as

$$D(B) = \min_{\pi \neq identity} D_\pi(B)$$

A graph B is symmetric if and only if $D(B) = 0$ [24]. We will next show that $D(B) > 0$ with high probability, for which we will define a few terms and prove some preliminary results. Let π be a permutation of vertices in V such that it fixes all but k vertices. Let Z be the set of vertices, $\{u|\pi(u) \neq u\}$ and

$$X = \sum_{u \in P} D_\pi(u)$$

Observe that, by definition, $D_\pi(u)$ is a binomially distributed random variable and $E[D_\pi(u)] = 2p(1-p)N$. Thus, $E[X] = 2p(1-p)kN$. Note that X depends only on the edges of the graph adjacent to the vertices in Z, and adding or deleting any such edge (u, v), for $u \in U$ and $v \in V$, will only affect $D_\pi(v)$ and $D_\pi(\pi^{-1}(v))$ each at most by 1. Since X is a sum of binomially distributed random variables, each of which is formed from mutually independent binary choices with some probability, it is a random variable formed from mutually independent probabilistic binary decisions, such that say with probability p_i it takes one of the two decisions. If the choices made for X can be indexed by i, and let c be a constant such that changing any such choice i would change X by at most c, then set $\sigma^2 = c^2 \sum_i p_i(1-p_i)$. In our case, $c = 2$, hence, $\sigma^2 = 4Nkp(1-p)$. For all positive $t < \frac{2\sigma}{c}$, it is shown in [44] that

$$P(|X - E[X]| > t\sigma) \leq 2e^{-\frac{t^2}{4}}$$

Set $\epsilon = \epsilon(N, p)$ such that $\epsilon = o(1)$ and $\epsilon^2 Np(1-p) \gg \ln N$. Then, for some positive constant α

$$\begin{aligned} & P(|X - E[X]| > \epsilon Nkp(1-p)) \leq 2e^{-\alpha\epsilon^2 Nkp(1-p)} \\ \implies & P(|X - E[X]| \leq \epsilon Nkp(1-p)) > 1 - 2e^{-\alpha\epsilon^2 Nkp(1-p)} \end{aligned}$$

Thus there exists a vertex u in Z such that $D_\pi(u) \geq \frac{(E[X] - \epsilon Nkp(1-p))}{k} = (2-\epsilon)Nkp(1-p)$ with probability at least $1 - 2e^{-\alpha\epsilon^2 Nkp(1-p)}$. Since, $D_\pi(B) = \max_v D_\pi(v)$, we have

$$P(D_\pi(B) \leq (2-\epsilon)Np(1-p)) \leq 2e^{-\alpha\epsilon^2 Nkp(1-p)}$$

Note that there are $\binom{N}{k}k!$ possible permutations such that $N-k$ vertices are fixed; thus, there exists a permutation π such that $D(B) < (2-\epsilon)Np(1-p)$ with probability less than

$$\sum_{k=2}^{N} \binom{N}{k} k! \times (2e^{-\alpha\epsilon^2 Nkp(1-p)})$$

As [24] shows, $\sum_{k=2}^{N} \binom{N}{k} k! \times (2e^{-\alpha\epsilon^2 Nkp(1-p)})$ is $O(N^{-w})$ for any positive constant w. Hence, a partially labeled random bipartite graph can be symmetric with probability at most $O(N^{-w})$. □

Lemma 18. *For all p satisfying $p \gg \frac{\ln N}{N}$ and $1-p \gg \frac{\ln N}{N}$, a random un-*

labeled bipartite graph is symmetric with probability $O(N^{-w})$ for any positive constant w.

Proof. Define $B = (\{U, V\}, E)$, an unlabeled bipartite graph with two sets of vertices U and V and set of edges E. Let $\pi : U \cup V \to U \cup V$ be the permutation of vertices in the sets U and V with constraints that $\pi(u) \in U$ if $u \in U$ and similarly $\pi(u) \in V$ if $u \in V$. Following the definitions of [34], for a vertex $v \in U \cup V$, we define a defect of v with respect to π to be

$$D_\pi(v) = |\Gamma(\pi(v))\Delta\pi(\Gamma(v))|$$

where $\Gamma(v)$ is the set of neighbors of v and Δ denotes the symmetric difference of two sets, i.e., $A\Delta B = (A - B) \cup (B - A)$ for two sets A and B. Similarly, one can define a defect of B with respect to π to be

$$D_\pi(B) = \max_v D_\pi(v)$$

and the defect of a graph B can be defined as

$$D(B) = \min_{\pi \neq identity} D_\pi(B)$$

A graph B is symmetric if and only if $D(B) = 0$ [24]. We will next show that $D(B) > 0$ with high probability, for which we will define a few terms and prove some preliminary results. Let π be a permutation of vertices in $U \cup V$ such that it fixes all but k vertices. Let Z be the set of vertices, $\{u|\pi(u) \neq u\}$ and

$$X = \sum_{u \in P} D_\pi(u)$$

Observe that, by definition, $D_\pi(u)$ is a binomially distributed random variable and $E[D_\pi(u)] = 2p(1-p)N$. Thus, $E[X] = 2p(1-p)kN$. Note that X depends only on the edges of the graph adjacent to the vertices in Z, and adding or deleting any such edge (u, v), for $u \in U$ and $v \in V$, will only affect $D_\pi(u)$, $D_\pi(\pi^{-1}(u))$, $D_\pi(v)$ and $D_\pi(\pi^{-1}(v))$ each at most by 1. Since X is a sum of binomially distributed random variables, each of which is formed from mutually independent binary choices with some probability, it is a random variable formed from mutually independent probabilistic binary decisions, such that say with probability p_i it takes one of the two decisions.

If the choices made for X can be indexed by i, and let c be a constant such that changing any such choice i would change X by at most c, then set $\sigma^2 = c^2 \sum_i p_i(1-p_i)$. In our case, $c = 4$, hence, $\sigma^2 = 16Nkp(1-p)$. For all positive $t < \frac{2\sigma}{c}$, it is shown in [44] that

$$P(|X - E[X]| > t\sigma) \leq 2e^{-\frac{t^2}{4}}$$

Set $\epsilon = \epsilon(N, p)$ such that $\epsilon = o(1)$ and $\epsilon^2 Np(1-p) \gg \ln N$. Then, for some positive constant α

$$\begin{aligned} & P(|X - E[X]| > \epsilon Nkp(1-p)) \leq 2e^{-\alpha\epsilon^2 Nkp(1-p)} \\ \implies & P(|X - E[X]| \leq \epsilon Nkp(1-p)) > 1 - 2e^{-\alpha\epsilon^2 Nkp(1-p)} \end{aligned}$$

Thus there exists a vertex u in Z such that $D_\pi(u) \geq \frac{(E[X] - \epsilon Nkp(1-p))}{k} = (2-\epsilon)Nkp(1-p)$ with probability at least $1 - 2e^{-\alpha\epsilon^2 Nkp(1-p)}$. Since, $D_\pi(B) = \max_v D_\pi(v)$, we have

$$P(D_\pi(B) \leq (2-\epsilon)Np(1-p)) \leq 2e^{-\alpha\epsilon^2 Nkp(1-p)}$$

Note that there are at most $\max_{k_1,k_2} \binom{N}{k_1}\binom{N}{k_2} k_1! k_2!$ possible permutations such that $k_1 + k_2 = k$ and $N - k$ vertices are fixed. Also, $\max_{k_1,k_2} \binom{N}{k_1}\binom{N}{k_2} k_1! k_2! \leq N^k$. Thus, there exists a permutation π such that $D(B) < (2-\epsilon)Np(1-p)$ with probability less than

$$\sum_{k=2}^{2N} N^k \times (2e^{-\alpha\epsilon^2 Nkp(1-p)}).$$

As [24] shows, $\sum_{k=2}^{N} N^k \times (2e^{-\alpha\epsilon^2 Nkp(1-p)})$ is $O(N^{-w})$ for any positive constant w. It follows that $\sum_{k=2}^{2N} N^k \times (2e^{-\alpha\epsilon^2 Nkp(1-p)})$ is also $O(N^{-w})$ for any positive constant w. Hence, an unlabeled random bipartite graph can be symmetric with probability at most $O(N^{-w})$. □

Lemma 19. *For all integers $n \geq 0$ and $d \geq 0$,*

$$a_{n,d}^2 \leq x_n$$

where x_n satisfies $x_0 = x_1 = 0$ and for $n \geq 2$

$$x_n = \lceil \log_2 (n+1) \rceil + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} (x_k + x_{n-k})$$

Proof. From Alg. 4, observe that $a^2_{0,d} = a^2_{1,d} = a^2_{2,0} = 0$. For $n \geq 2$, observe the following recursion relations for $a^2_{n,d}$:

$$a^2_{n+1,0} = \lceil \log_2 (n+1) \rceil + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} (a^2_{k,1} + a^2_{n-k,2k+1})$$

$$a^2_{n,d} = \lceil \log_2 (n+1) \rceil + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} (a^2_{k,d-1} + a^2_{n-k,2k+d-1})$$

We will prove the lemma using induction on both n and d. For the base cases, observe that for $n = 0$ or 1, $a^2_{n,d} \leq x_n$. Further, for $n = 2$ and $d = 0$, $a^2_{2,0} \leq x_2$. Now, assuming that $a^2_{i,j} \leq x_i$ for $i < n$, and for $i = n$ and $j < d$, we want to show that $a^2_{n,d} \leq x_n$. We will consider the following two cases.

Case $d = 0$: From the recursion relation of x_n it follows that $x_n = \lceil \log (n+1) \rceil + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (x_k + x_{n-k}) + (p^n + q^n)(\sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (x_k + x_{n-k})) + (p^n + q^n)^2 (x_n)$, which implies that, $x_n (1 - (p^n + q^n)^2) = \lceil \log (n+1) \rceil + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (x_k + x_{n-k}) + (p^n + q^n)(\sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (x_k + x_{n-k}))$.

Similarly, $a^2_{n,0} \leq a^2_{n+1,0} = \lceil \log (n+1) \rceil + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} (a^2_{k,1} + a^2_{n-k,2k+1})$ implies that, $a^2_{n,0} \leq \lceil \log (n+1) \rceil + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (a^2_{k,1} + a^2_{n-k,2k+1}) + (p^n + q^n)(a^2_{n,1})$ which in turn implies that, $a^2_{n,0} \leq \lceil \log (n+1) \rceil + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (a^2_{k,1} + a^2_{n-k,2k+1}) + (p^n + q^n)(\sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (a^2_{k,0} + a^2_{n-k,2k})) + (p^n + q^n)^2 (a^2_{n,0})$, which yields

$$a^2_{n,0}(1 - (p^n + q^n)^2) \leq \lceil \log (n+1) \rceil + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (a^2_{k,1} + a^2_{n-k,2k+1})$$
$$+ (p^n + q^n)(\sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (a^2_{k,0} + a^2_{n-k,2k}))$$

Further,

$$a^2_{n,0}(1-(p^n+q^n)^2) \le \lceil \log{(n+1)} \rceil + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k}(x_k + x_{n-k})$$
$$+ (p^n+q^n)(\sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k}(x_k + x_{n-k}))$$

implies that

$$a^2_{n,0}(1-(p^n+q^n)^2) \le \lceil \log{(n+1)} \rceil + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k}(x_k + x_{n-k})$$
$$+ (p^n+q^n)(\sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k}(x_k + x_{n-k}))$$

which implies that $a^2_{n,0} \times (1-(p^n+q^n)^2) \le x_n \times (1-(p^n+q^n)^2)$.

Case $d > 0$: $a^2_{n,d} = \lceil \log_2{(n+1)} \rceil + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k}(a^2_{k,d-1} + a^2_{n-k,2k+d-1})$ implies that $a^2_{n,d} \le \lceil \log_2{(n+1)} \rceil + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k}(x_k + x_{n-k})$ which yields $a^2_{n,d} \le x_n$. □

Lemma 20. *For all $n \ge 0$ and $d \ge 0$,*

$$b^2_{n,d} \ge y_n - \frac{n}{2},$$

such that y_n satisfies $y_0 = 0$ and for $n \ge 0$,

$$y_{n+1} = n + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k}(y_k + y_{n-k})$$

Proof. First observe that $b_{0,d} = b_{1,d} = b_{2,0} = 0$, and for $n \ge 2$, $b^2_{n,d}$ forms the following recursion relation.

$$b^2_{n+1,0} = n + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k}(b^2_{k,1} + b^2_{n-k,2k+1})$$
$$b^2_{n,d} = n + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k}(b^2_{k,d-1} + b^2_{n-k,2k+d-1})$$

We will use induction on both n and d to prove the claim. For the base cases, clearly for $n = 0$ or $n = 1$, $b^2_{n,d} \geq y_n - \frac{n}{2}$. Also for $n = 2$ and $d = 0$, $b^2_{n,d} \geq y_n - \frac{n}{2}$ holds. Now, assuming that $b^2_{i,j} \geq y_i - \frac{i}{2}$ for $i < n$, and for $i = n$ and $j < d$, we want to show that $b^2_{i,j} \geq y_i - \frac{i}{2}$. We will consider the following two cases.

Case $d = 0$: $b^2_{n,0} = (n-1) + \sum_{k=0}^{n-1} \binom{n-1}{k} p^k q^{n-k-1} (b^2_{k,1} + b_{n-k-1,2k+1})$ which implies $b^2_{n,0} \geq (n-1) + \sum_{k=0}^{n-1} \binom{n-1}{k} p^k q^{n-k-1} (y_k - \frac{k}{2} + y_{n-k-1} - \frac{n-k-1}{2})$ that leads to $b^2_{n,0} \geq y_n - \frac{n-1}{2}$ and finally, $b^2_{n,0} \geq y_n - \frac{n}{2}$.

Case $d > 0$: $b^2_{n,d} = n + \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} (b_{k,d-1} + b^2_{n-k,2k+d-1})$ implies $b^2_{n,d} \geq y_{n+1} - \frac{n}{2}$. From [24], we know that $y_{n+1} \geq y_n$, and so $b^2_{n,d} \geq y_n - \frac{n}{2}$. $\square$

REFERENCES

[1] 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.

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

[3] M. Patrascu, "Succinter," in *Proc. 49th Annu. IEEE Symp. Found. Comput. Sci.*, Oct. 2008, pp. 305–313.

[4] M. Mitzenmacher, "Compressed Bloom filters," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 604–612, Oct. 2002.

[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)*, July 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)*, July 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] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," arXiv:1503.02531v1 [stat.ML]., Mar. 2015.

[15] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 126–136, Jan. 2018.

[16] D. B. Chklovskii, B. W. Mel, and K. Svoboda, "Cortical rewiring and information storage," *Nature*, vol. 431, no. 7010, pp. 782–788, Oct. 2004.

[17] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1," arXiv:1602.02830 [cs.LG]., Feb. 2016.

[18] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights," in *Proc. 2016 IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, July 2016, pp. 236–241.

[19] L. R. Varshney, P. J. Sjöström, and D. B. Chklovskii, "Optimal information storage in noisy synapses under resource constraints," *Neuron*, vol. 52, no. 3, pp. 409–423, Nov. 2006.

[20] P. Khadivi, R. Tandon, and N. Ramakrishnan, "Flow of information in feed-forward deep neural networks," arXiv:1603.06220 [cs.IT]., Mar. 2016.

[21] L. R. Varshney and V. K. Goyal, "Toward a source coding theory for sets," in *Proc. IEEE Data Compression Conf. (DCC 2006)*, Mar. 2006, pp. 13–22.

[22] Y. A. Reznik, "Coding of sets of words," in *Proc. IEEE Data Compression Conf. (DCC 2011)*, Mar. 2011, pp. 43–52.

[23] C. Steinruecken, "Compressing sets and multisets of sequences," *IEEE Trans. Inf. Theory*, vol. 61, no. 3, pp. 1485–1490, Mar. 2015.

[24] Y. Choi and W. Szpankowski, "Compression of graphical structures: Fundamental limits, algorithms, and experiments," *IEEE Trans. Inf. Theory*, vol. 58, no. 2, pp. 620–638, Feb. 2012.

[25] J. J. Rissanen, "Generalized Kraft inequality and arithmetic coding," *IBM J. Res. Develop.*, vol. 20, no. 3, pp. 198–203, May 1976.

[26] J. Rissanen and G. G. Langdon, Jr., "Arithmetic coding," *IBM J. Res. Develop.*, vol. 23, no. 2, pp. 149–162, Mar. 1979.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[28] T. Cohen and M. Welling, "Group equivariant convolutional networks," in *Proc. 33rd Int. Conf. Mach. Learn. (ICML 2016)*, June 2016, pp. 2990–2999.

[29] R. Kondor and S. Trivedi, "On the generalization of equivariance and convolution in neural networks to the action of compact groups," in *Proc. 35th Int. Conf. Mach. Learn. (ICML 2018)*, July 2018, pp. 2747–2755.

[30] S. Ravanbakhsh, J. Schneider, and B. Póczos, "Equivariance through parameter-sharing," in *Proc. 34th Int. Conf. Mach. Learn. (ICML 2017)*, Aug. 2017, pp. 2892–2901.

[31] Z. Wang, W. Zou, and C. Xu, "Pr product: A substitute for inner product in neural networks," in *Proc. 23rd IEEE Int. Conf. Computer Vision*, Oct. 2019, pp. 6013–6022.

[32] F. Harary and E. M. Palmer, *Graphical Enumeration.* New York: Academic Press, 1973.

[33] F. Harary, E. M. Palmer, and R. C. Read, "The number of ways to label a structure," *Psychometrika*, vol. 32, no. 2, pp. 155–156, June 1967.

[34] J. H. Kim, B. Sudakov, and V. H. Vu, "On the asymmetry of random regular graphs and random graphs," *Random Struct. Algorithms*, vol. 21, no. 3-4, pp. 216–224, Oct.-Dec. 2002.

[35] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. New York: John Wiley & Sons, 2006.

[36] P. Elias, “Universal codeword sets and representations of the integers,” *IEEE Trans. Inf. Theory*, vol. 21, no. 2, pp. 194–203, 1975.

[37] Y. LeCun, C. Cortes, and C. J. C. Burges, “The MNIST database of handwritten digits,” 2018, http://yann.lecun.com/exdb/mnist/.

[38] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, “Learning word vectors for sentiment analysis,” in *Proc. Assoc. Comput. Linguist. Annu. Meet. (ACL 2011)*, June 2011, pp. 142–150.

[39] D. Lewis, “Reuters-21578 text categorization test collection,” *Distribution 1.0, AT&T Labs-Research*, 1997.

[40] 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.

[41] 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.

[42] W. Szpankowski, “Algorithms, combinatorics, information, and beyond,” *IEEE Inf. Theory Soc. Newsletter*, vol. 62, no. 6, pp. 5–20, June 2012.

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

[44] N. Alon, J.-H. Kim, and J. Spencer, “Nearly perfect matchings in regular simple hypergraphs,” *Israel J. Math.*, vol. 100, no. 1, pp. 171–187, Dec. 1997.