A GRAPH NETWORK MODEL FOR DISTRIBUTED LEARNING WITH LIMITED BANDWIDTH LINKS AND PRIVACY CONSTRAINTS

Juan Parras, Santiago Zazo

Information Processing and Telecommunications Center
Universidad Politécnica de Madrid
Madrid, Spain

ABSTRACT

In this work, we develop an algorithm based on graph networks to train distributedly a deep learning model. We consider that there are several nodes, in an arbitrary network topology, each one of them having access to a local dataset that, for privacy concerns, cannot be shared with other nodes. We also assume that there are bandwidth constraints, and hence, it may not be affordable sharing the weights of the model with other nodes. Our algorithm makes use of pruning and an autoencoder to train under all these constraints, and our simulations show that it provides a good accuracy, while preserving the privacy of the data and providing a compression of nearly two orders of magnitude in the transmitted bits.

Index Terms— Deep Learning, Distributed Systems, Autoencoder, Graph Networks

1. INTRODUCTION

Privacy is becoming a very important topic in our world today, and a lot of research is devoted to this area, as a recent survey shows [1]. In this work, we are interested in training a Deep Learning (DL) model, in which the data is distributed among different agents communicated among them. If the local dataset of all agents are identically distributed, they are balanced, and unbalanced otherwise. We should not share the training dataset among the agents for privacy reasons [2]. Although there are techniques to preserve privacy while doing a centralized training [3], it is more frequent to work distributedly when privacy matters [4], [5].

As shown in [6], a lot of research has been dedicated to obtaining algorithms for training DL models in a parallel / distributed fashion [6]. However, we are interested in developing an algorithm that simultaneously (1) deals with unbalanced local datasets, (2) adapts to bandwidth restrictions (3) preserves the privacy of the local datasets and (4) is fully distributed. Most current algorithms satisfy only some of these four assumptions: Stochastic Gradient Descent approaches, both synchronous and asynchronous, are communication intensive: in each iteration, they need to transmit a gradient vector which, in case of a DL model, may have millions of components [7], [8], [9]. As an alternative to synchronous and asynchronous SGD methods, models based on diffusion mechanisms are being proposed as a way to overcome the limits imposed by the communication links, as Gossiping SGD [7], which however, does not scale well, or GossipGradD [10], which requires balanced datasets.

The work is supported by the Spanish Ministry of Science and Innovation under the grant TEC2016-76038-C3-1-R (HERAKLES). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan V GPU used for this research.

Moreover, as shown in [6], most algorithms for training DL models in a parallel / distributed fashion preserve a copy of the model weights which are visible to all agents. This assumption needs not hold in distributed settings, as different nodes with different datasets will have different weights in their local DL models. It is possible either to train locally several DL models and provide as output an average of these models [11] or combining model weights during training [12], which is the approach that we choose to use.

However, as we noted, exchanging the model weights or the gradient of these weights may be unfeasible if bandwidth is limited, as in mobile systems, where DL models are becoming subjects of interest [13]. Hence, compressing a DL model has been also the focus of recent research. It is possible to compress a DL model by taking advantage of the redundancies of the weights [14]. A different way consists in using pruning: current DL models are oversized and hence, there are small subsets of weights which are enough to provide a good accuracy in DL models [15]. This is used in [5], where only the weights with larger gradient norm are transmitted. In [16] and [15], the weights that are transmitted are those with larger norm. We make use of pruning also for our algorithm.

Finally, we model the communication network using the Graph Network model [17], which generalizes many prior works which used graphs and neural networks in order to deal with data while preserving the relations among the data. One of the firsts works is [18], where Graph Neural Networks are proposed and its convergence properties are discussed. Many other works followed, such as [19], [20] or [21], to mention some. It is an expanding field, whose applications range from image and text processing, to molecular applications in biology, social network interactions or multi-agent systems [22], [23]. We choose to use the Graph Network model [17] because it generalizes many of these previous graph based structures and provides a convenient framework to analyze our problem.

The rest of the paper goes as follows: Section 2 introduces the mathematical concepts that are needed to understand our problem, which is presented in Section 3. Our proposed algorithm is presented in Section 4, and then we validate our approach in Section 5. We finally draw some conclusions in Section 6.

2. BACKGROUND

2.1. Graph Networks

Graph Networks (GNs) [17] are a general artificial intelligence framework that makes use of graphs in order to learn the different relations among the available data in a structured way. Let us define a graph triple $G = (u, V, E)$, where u is a vector representing the global state of the graph, $V = \{v_m\}_{m=1:N_v}$ is the set of nodes of the graph, where we have N_v nodes and each v_m is a vector

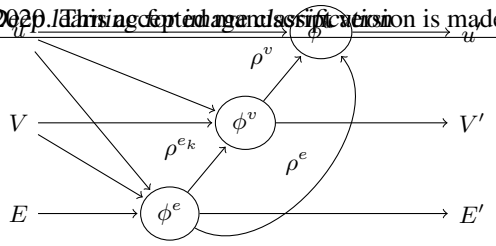


Fig. 1. GN structure, where we see how each of the elements of the input graph $G = (u, V, E)$ is processed using the functions ϕ^u, ϕ^v, ϕ^e, ρ^{e_k}, ρ^e and ρ^v to obtain the output graph $G' = (u', V', E')$.

containing the attributes of node m, and $E = \{e_k, r_k, s_k\}_{k=1:N_e}$ is the set of edges of the graph, where e_k is a vector containing the attributes of the edge k, s_k is the index of the sender node and r_k is the index of the receiver node.

The GN model defines the GN block as a graph to graph mapping: a graph G is provided as input and the graph G' is returned as output using the following equations:

$$\begin{aligned} e'_k &= \phi^e\left(e_k, v_{r_k}, v_{s_k}, u\right), \\ v'_m &= \phi^v\left(\rho^{e_k}\left(e'_k\right), v_m, u\right), \\ u' &= \phi^u\left(\rho^v\left(v'_m\right), \rho^e\left(e'_k\right), u\right), \end{aligned} \tag{1}$$

where ϕ^e, ϕ^v and ϕ^u are functions that allow updating the edges, nodes and global state, respectively, as illustrated in Figure 1. These functions are the same for all nodes and edges of the graph, and may be implemented using neural networks. The ρ functions are called diffusion functions: they aggregate data from several nodes or edges to generate a combined output: these functions need to be independent to the number of inputs and their ordering, hence, it is frequent using the sum function, the maximum, the minimum or, as we do, randomly choosing one input and transmit it.

In the output graph G', the attributes of each edge e'_k are obtained by updating the input graph edges using information about the global state, as well as the pair of nodes that are connected by each edge. The attributes of each node in the output graph, v'_m, are obtained using the attributes of the node and the global state from the input graph, v_m and u, and the aggregated information from the edges that have node m as receiver. Finally, the output global state is obtained with the function ϕ^u, which takes as inputs all the attributes of the nodes and edges from the output graph, as well as the input global state.

2.2. Deep learning for image classification

DL has been particularly successful for image processing tasks, such as classification or labeling [24]. One of the keys for this success has been the use of Convolutional Neural Networks (CNNs). CNNs are designed to extract information from local environments, and hence, they take advantage of the strong spatial correlation in images [25]. They have become the standard tool for image analysis today.

2.3. Autoencoders

Autoencoders are a special type of neural network, which is trained so that the output is the same vector as the input [25]. It usually has two parts: the encoder and the decoder. The encoder takes the input vector and produces as output a latent vector, which is usually smaller than the input vector. The decoder takes as input the latent vector and produces as output a vector which tries to match the input. Hence, the main idea behind autoencoders is being able to extract meaningful features that allow compressing the input data to a latent vector representation with a smaller dimension.

3. PROBLEM DESCRIPTION

We consider a problem in which we have M agents, each one of them with a private local dataset D_m, $m \in 1, 2, ..., M$, where the set D_m of each agent has N_m tuples of data x and labels y. Thus, $D_m = \{(x_{1,m}, y_{1,m}), (x_{2,m}, y_{2,m}), ..., (x_{N_m,m}, y_{N_m,m})\}$. We define $D = \cup_m D_m$ as the total dataset. The target of each agent is to train a DL model, with weights θ_m, such that it is able to approximate the function $f_{\theta_m}(x_{i,m}) = y_{i,m}$ by minimizing a certain loss function $l(y_{i,m}, f_\theta(x_{i,m}))$, $\forall(x_{i,m}, y_{i,m}) \in D$. Note that we want to minimize the loss function with respect to the total dataset D, however, each agent m has only access to a certain subset D_m of D. Also, note that we assume that all agents have the same function f, i.e., that they share the same DL model. Each agent can communicate with the agents in its neighborhood, and the network has an arbitrary topology. As we want to preserve the privacy of each agent, agent m cannot share its dataset D_m. Thus, each agent can only transmit h_m, a vector of information to other agents such that it gives them useful information about its own dataset. At the same time, h_m has to be bounded, as we consider that we have a limited transmission bandwidth among agents.

4. PROPOSED ALGORITHM

We now describe Graph AutoEncoder (GAE), the solution that we propose for the problem described in Section 3.

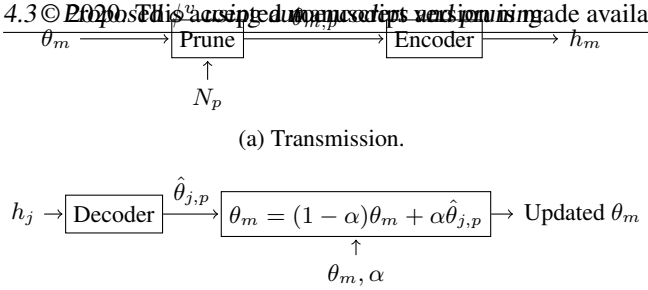
4.1. Graph formulation of the problem

We make use of the GN formulation in order to define our problem. We use the graph $G = (u, V, E)$ to model the network formed by the agents, where each agent is a node of the graph, hence, $M = N_v$. We consider that the attributes of node m are the concatenation of h_m and its local losses $l_m = l\left(y_{i,m}, f_{\theta_m}(x_{i,m})\right), \forall(x_{i,m}, y_{i,m}) \in D_m$. Hence, $v_m = [h_m, l_m]$, where $[\ \ ]$ represents the concatenation operator. The edges of the graph represent the communication capabilities of each agent, that is, the edge e_k, r_k, s_k means that the agent s_k can communicate with agent r_k. We consider that each edge represents a unidirectional communication link. The attribute of each edge is its bandwidth. The global state u represents the average of the loss function values, $u = \frac{1}{M}\sum_m l_m$. We consider that the diffusion functions ρ consists in randomly selecting one agent of the neighborhood and exchanging information with it. We only need to define each the functions ϕ^e, ϕ^v and ϕ^u.

4.2. Definition of ϕ^v

The key part in our model is the function ϕ^v, by which each node m updates its attributes $v_m = [h_m, l_m]$. As each node m has only access to D_m, node m needs to send useful information h_m about its model f_{θ_m} to the nodes in its neighborhood, and receive information from the other nodes. A possible solution proposed in the literature to the problem we are addressing is simply to transmit the model weights, that is, $h_m = \theta_m$ [12]. If node m receives the weights from node j, then node m updates its weights as:

$$\theta_m = (1 - \alpha)\theta_m + \alpha\theta_j, \tag{2}$$



(a) Transmission.

(b) Reception.

Fig. 2. Autoencoder structure for dimensionality reduction. The encoder obtains the w values with largest positive magnitude, the w weights with largest negative magnitude, encodes the rest of weights using the encoder part of the autoencoder and transmits this information, as well as the pruning threshold. All that information is used by the decoder in order to reconstruct the weights.

where $\alpha \in [0, 1]$ is a parameter that controls the update of the weights. As diffusion function ρ^{e_k} for node m, as said before, we randomly choose one of the potentially many edges that point towards node m, and takes its h vector in order to update the weights θ_m. Thus, under this paradigm, the function ϕ^v becomes:

$$\phi^v = [h_m, l_m]\,. \tag{3}$$

Note that we assume that there is no error in the communication of the weights on the network, although we do not enter into details of the communication mechanism.

4.3. Proposed ϕ^v using autoencoders and pruning

Current DL models can have millions of parameters [24], thus, the amount of data to transmit may be too large for limited bandwidth links. Note that the bits required to transmit θ_m is b_θ:

$$b_\theta = N_\theta b_{float}, \tag{4}$$

where N_θ is the number of weights and b_{float} the bits used to encode a float number. A possible way to reduce the bits to transmit is to combine pruning with quantization and Huffman coding, as in [16], but this method is computationally intensive, as the quantization centroids have to be calculated for each transmission.

We propose reducing the bits to transmit by using pruning and an autoendoder structure, as shown in Figure 2. When node m wants to obtain h_m, it starts by obtaining its weights θ_m and pruning them to obtain the pruned version of the weights, $\theta_{m,p}$. The pruning is done by selecting the N_p weights with larger norm 2 and setting the rest of weights to zero [26]. The resulting vector of weights is sparse, and thus, we need transmitting only the non-zero values as float numbers and their indexes as integers. Pruning transmits $b_{\theta,pr}$ bits:

$$b_{\theta,pr} = N_p\left(b_{int} + b_{float}\right), \tag{5}$$

where b_{int} is the number of bits used to encode an integer number.

In order to further reduce the dimension of h_m, we also use an autoencoder structure. The sender node has the encoder part of the autoencoder: it provides as input $\theta_{m,p}$ and encodes the weights as follows. We transmit the $2w$ largest norm 2 weights of $\theta_{m,p}$, as these are the most important, and use the autoencoder to encode the rest of weights. We also transmit the pruning threshold, so that the decoder can set to 0 all values below that threshold in order to account for error in the autoencoder reconstruction. Thus, the encoder sends h_m, composed of $2w$ weights, the latent vector of the autoencoder and the indexes of the weights $\theta_{m,p}$. The receiver node needs to have the decoder part of the autoencoder, in order to obtain $\hat{\theta}_{m,p}$, the reconstruction of $\theta_{m,p}$, from the information in h_m. The reconstructed weights are combined using (2). Note that in this case, we need to transmit the N_p indexes as integers, but now, instead of transmitting N_p weights, we transmit only $N_l + 2w$ weights and the pruning threshold, where $N_l < N_p$ is the latent dimension of the autoencoder. Thus, the number of bits transmitted now are:

$$b_{\theta,pr,ae} = N_p b_{int} + \left(2w + N_l + 1\right) b_{float}, \tag{6}$$

where we note that this method requires that all nodes have the same encoder and decoder models, which are trained a priori and distributed to all nodes of the network.

4.4. Definition of ϕ^e and ϕ^u

In our problem, the edges are static, that is, we do not change the edges attribute, which is its bandwidth b_k. We use the edges to send the node losses and the node information vector h. Thus:

$$\phi^e\left(e_k, v_{s_k}\right) = \left[b_k, v_{s_k}\right] = \left[b_k, h_{s_k}, l_{s_k}\right]. \tag{7}$$

Note that the bandwidth in the edges limits the maximum number of bits that they can transmit. This is a limitation that needs to be taken into account when designing the compression mechanism, that is, the parameters N_p and N_l (6).

Finally, the global state is the mean of the losses of each agent, and hence, by denoting l'_m to the current losses of node m:

$$\phi^u\left(\sum_m v'_m\right) = \frac{1}{M}\sum_m l'_m. \tag{8}$$

4.5. GAE algorithm overview

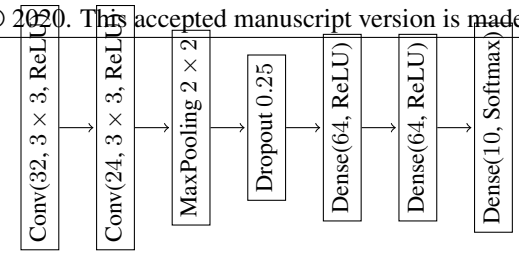
Algorithm 1 Overview of GAE.

Input: The network connections V, the list of m nodes with their D_m, T, f, autoencoder networks, α and N_p.

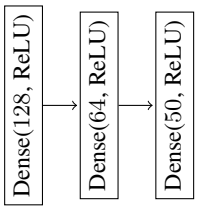
1: Initialize E, setting h_m and l_m to zero; and $u = +\infty$.
2: Obtain $G = (u, V, E)$, the graph of the network.
3: **for** $t \in 1, 2, ..., T$ **do**
4: Each node m trains its local model f_{θ_m} by using its local database D_m, obtaining l_m.
5: Each node m obtains h_m.
6: Update e'_k using (7), v'_k using (3) and u' using (8).

Output: M models f_{θ_m} trained distributedly on D.

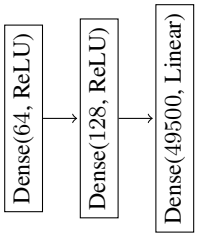
The whole GAE procedure can be seen in Algorithm 1, which integrates all the parts described in the previous sections. First, we have to set α and N_p, as well as provide the autoencoder model to each node. Then, for a certain number of iterations T, for each node m, GAE alternatively trains f_{θ_m} on the local dataset D_m, and then communicates to other nodes and updates the parameters θ_m. Note that this process needs not be synchronous, as each node may train and diffuse its information in different times. The update process of ϕ^v has been already explained in Section 4.3.



(a) Local DL model f_{θ_m}.



(b) Encoder.



(c) Decoder.

Fig. 3. DL models used in our experiments. Conv(a, b, c) stands for a convolutional layer with a neurons, b kernel and c as activation function, and Dense(a, c) is similar for a dense layer. We use Rectified Linear Units activations (ReLU) in all layers except the final one of the local model, where a softmax activation function is used in order to give a probability, and the final layer of the decoder, as the output data may take any value in $(-\infty, +\infty)$.

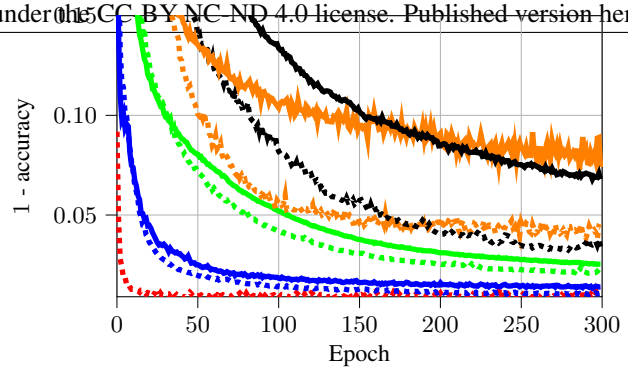


Fig. 4. Results obtained: red is the centralized case, blue is for 4 nodes and balanced datasets, orange is for 4 nodes unbalanced, green is for 16 nodes balanced, and black is for 16 nodes unbalanced. Baselines are dotted, and our proposed algorithm, GAE, in solid lines, produces similar accuracies to the baselines.

	Centralized	4 nodes	16 nodes
GAE balanced		0.9861	0.9747
Baseline balanced	0.9904	0.9893	0.9780
GAE unbalanced		0.9260	0.9320
Baseline unbalanced		0.9562	0.9661

Table 1. Accuracy obtained in validation.

5. RESULTS

Now, we show the results for different setups in which we test and validate GAE. We focus on studying the effects of our proposed architectures on the accuracy and the bits needed for transmission. We consider a ring topology network, in which there are $M = \{1, 4, 16\}$ nodes, where $M = 1$ is the centralized case, i.e., a single node has access to D.

We test GAE using the MNIST digits database, which is a well-known benchmark [27]. It has 60000 28×28 training grayscale images with handwritten digits between 0 and 9, with other 10000 images for validation. Each image comes labeled with the digit in the image. We split the training set in M parts to form the D_m sets, and validate each local model using the 10000 validation images. We propose two different settings: one in which D_m contains images from all digits, i.e., balanced datasets, and another in which each D_m contains samples from 8 randomly assigned digits, i.e., unbalanced datasets. The local DL model used to classify the MNIST digits can be seen in Figure 3. We train using Adam [28].

The architecture that we use for the autoencoder can be seen in Figure 3. In our case, we trained our autoencoder using 5000 / 500 weights for training/validation, where the weights were obtained from training the local model using the centralized scheme. The performance of the autoencoder increases significantly if the input data is sorted, as this adds correlation; note that, since we have to transmit the data indexes, this does not increase the amount of data to transmit. After 25 training epochs using Adam and minimum square error as loss function, the autoencoder converged. The same autoencoder has been used for all our experiments.

Finally, as parameters for our autoencoder architecture, we use $\alpha = 0.5$, $N_p = 50000$, $w = 250$ and $N_l = 50$. As the local DL model has $N_\theta = 613514$ parameters, we are pruning by a factor of roughly 10%. We train the local models for $T = 300$ epochs, following the GAE description from Algorithm 1 and using the categorical crossentropy as loss function. We initialize all f_{θ_m} with the same weights, as pruning significantly improves using this [15]. As baseline, we compare with the case in which we transmit the pruned weights. The results obtained can be observed in Figure 4 and Table 1. We note that the highest accuracy occurs in the centralized case, and it decreases with the number of nodes. Also, note that the accuracy in the unbalanced case is lower than in the balanced case, as expected. Significantly, note that GAE produces a low decrease in accuracy.

Considering that $b_{float} = 32$ and $b_{int} = 20$, as we need 20 bits to transmit the positions of the 613514 weights, we can compare the bits required for each case. Without compression, we have $b_\theta = 19.63 \cdot 10^6$ (4). With pruning, we have $b_{\theta,pr} = 2.6 \cdot 10^6$ (5). Using GAE we have $b_{\theta,pr,ae} = 1.02 \cdot 10^6$ (6). Note how using an autoencoder as we propose helps both significantly compressing the weights in nearly two orders of magnitude compared to not using pruning, and in more than half to using pruning, while preserving a good accuracy that is always above 90% and similar to the pruning baseline in all situations, as the accuracy decrease is around 3% in the worst case. Hence, GAE is successful: it reduces the data transmitted while preserving a good accuracy.

6. CONCLUSIONS

GAE is a promising algorithm that may be extended in several directions, such as providing convergence analysis or checking its properties for a wider set of parameters. This is a first step towards distributed algorithms to train DL models that preserve the privacy and operate under limited bandwidth, which may have many applications in the current world.

7. REFERENCES

[1] Isabel Wagner and David Eckhoff, "Technical privacy metrics: a systematic survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 57, 2018.

[2] Natalia Neverova, Christian Wolf, Griffin Lacey, Lex Fridman, Deepak Chandra, Brandon Barbello, and Graham Taylor, "Learning human identity from motion patterns," *IEEE Access*, vol. 4, pp. 1810–1820, 2016.

[3] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 308–318.

[4] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[5] Reza Shokri and Vitaly Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 2015, pp. 1310–1321.

[6] Tal Ben-Nun and Torsten Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 65, 2019.

[7] Peter H Jin, Qiaochu Yuan, Forrest Iandola, and Kurt Keutzer, "How to scale distributed deep learning?," *arXiv preprint arXiv:1611.04581*, 2016.

[8] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, 2011, pp. 693–701.

[9] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al., "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[10] Jeff Daily, Abhinav Vishnu, Charles Siegel, Thomas Warfel, and Vinay Amatya, "Gossipgrad: Scalable deep learning using gossip communication based asynchronous gradient descent," *arXiv preprint arXiv:1803.05880*, 2018.

[11] Kai Chen and Qiang Huo, "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," in *2016 ieee international conference on acoustics, speech and signal processing (icassp)*. IEEE, 2016, pp. 5880–5884.

[12] Yajie Miao, Hao Zhang, and Florian Metze, "Distributed learning of multilingual dnn feature extractors using gpus," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014, pp. 830–834.

[13] Chaoyun Zhang, Paul Patras, and Hamed Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys & Tutorials*, 2019.

[14] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, 2015, pp. 2285–2294.

[15] Jonathan Frankle and Michael Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[16] Song Han, Huizi Mao, and William J Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[17] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al., "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[18] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[19] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[20] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel, "Neural relational inference for interacting systems," in *International Conference on Machine Learning*, 2018, pp. 2693–2702.

[21] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1263–1272.

[22] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[23] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[24] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[25] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436, 2015.

[26] Song Han, Jeff Pool, John Tran, and William Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[28] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations*, 2015.