Hardware-aware Distributed Learning Algorithm

Ching-Yi Lin Wei Chen

Abstract

As state-of-art approaches for non-convex Federated Learning, Federated Averaging and FedMAX contribute to multiple applications due to their communication-efficiency. However, there is a sporadic report of the algorithm implementation on hardware. In this work, we utilize the communication-efficient algorithms into the edge devices (Raspberry Pi). By comparing the communication cost of different network structures, we propose a hierarchical network to improve the communication-efficiency further. We measure the test accuracy and the communication latency of our network. We also simulate the model compression to handle the device heterogeneity where edge devices can have varied computational resources.

1. Introduction

Millions of data are generated from edge devices, such as cell phones and tablets. It is appealing to train an intelligent model based on these data. However, with privacy concerns, cellphone users tend to preserve their data on their own devices instead of sharing it with the server. Federated learning (FL)[6], in which models are trained locally on each client[1] and merged into a global model on the server, becomes a new paradigm to train the distributed data.

In real scenarios, there exists statistical and system heterogeneity in the FL setting.

1. Statistical heterogeneity: Distributed learning assumes the independent and identically distributed (IID) data across devices. However, since the data are collected from different users, the data distribution will fluctuate due to personal preference.

2. System heterogeneity: The devices involved in FL training have different hardware configurations, such as different computational power, communication speed.

[1]Clients are also called devices, nodes, or users in the FL setting.

This project targets the system heterogeneity. Specifically, we exploit the hierarchical structure of the network to achieve a communication-efficient FL system.

Changing topology is not a simple implementation problem. In network science or computer system, more local computation usually implies higher efficiency or more exceptional performance. However, in machine learning, less communication between data suggests using less amount of data during the training, which might introduce less generalizability or lower accuracy; even worse, the training process overfits the split dataset.

As a communication efficient algorithm, FedMAX[5] can effectively deal with non-identically distributed (non-IID) data. However, there is no result in the communication latency of FedMAX in a real application. It is also unknown whether FedMAX can be suitable for devices with varied computation resources.

In our work, we implement FedAvg and FedMAX in a Raspberry Pi network and evaluate the performance of different network topology. The remainder of this paper is organized as follows. In section 2, we provide background on federated learning and an overview of related work. In section 3, we discuss the assumptions in the FL scenario and describe our approaches and algorithms. Then we present our experimental setting and provide a thorough empirical evaluation of the proposed algorithm in section 4. Our empirical results help to illustrate and validate the communication-efficient of the algorithms in a hardware system and demonstrate the practical improvements of the hierarchical network over the star network.

2. Previous work

In the non-convex setting, Federated Averaging (FedAvg), based on averaging weight updates from each node in the central server, has been shown to work well empirically[7]. Unfortunately, FedAvg is quite challenging to deal with the issue that data is heterogeneously distributed in the network. Moreover, as each device generates its local data, statistical heterogeneity is common when data being non-IID between devices.

As a communication efficient algorithm, FedMAX[5] can deal with non-identical distributed (non-IID) data.

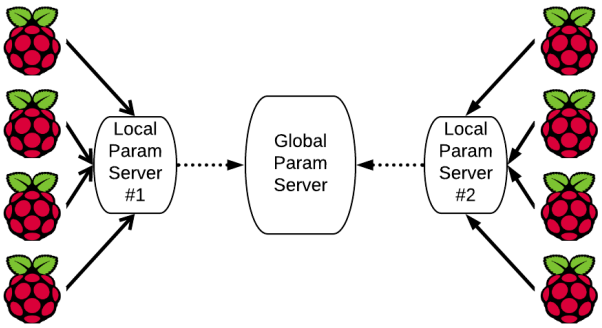

Figure 1. Hierarchical node network. The raspberry pi devices are connected to the local server through the wired network, which has shorter communication time, and local servers are connected to the global server through the wireless network, which has longer communication time.

Compared with FedAvg, FedMAX has an additional cost term in the object function, which acts as a constraint to prevent the clients from varied too far from each other. Nevertheless, the FedMAX is lack of implementation and verification of its communication efficiency.

In practical FL, systems contain servers and clients, and there is a network protocol[1] dealing with this setting. The protocol requires a central server and multiple devices forming a star network, and models from different devices are updated synchronously. Furthermore, each time only a small part (about 10%) of total devices can be involved in the training process. In this case, the model cannot be efficiently updated, and many available devices waste time waiting for getting involved in the training. So it is necessary to develop a hierarchical structure network so that more devices can contribute to the training.

3. Approach

3.1 Network architecture

Communication time is profoundly affected by traffic, bandwidth, and distance. In this project, we explore the two-layer hierarchical network. The structure of the proposed network is in figure 1; the network consists of a central global server, multiple local servers, and a large amount of compute nodes connected to their local server. We will show the benefit of this efficient network structure in the following part.

3.1.1 Out-degree of nodes

The point-out edge shows the model update path for a node, as shown in Fig.1. Compared to an arbitrary network, in which out-degree for a node can be more than 1, the updating policy should be involved in to choose between all the server candidates. In our network, the out-degree-1 property uniquely specifies the update path for each node. This property appears in a tree-based network usually and gives us a static graph during the runtime.

3.1.2 In-degree of nodes

Due to the heterogeneous communication cost, the local servers act as mailmen collecting all the updated results. Then send the merged result in a single packet. This method can get benefit from a node with large in-degree since it prevents all the leaf nodes from spending an expensive cost to communicate with the faraway node.

3.2 Assumption

1. **Non-i.i.d. data distribution:** The training data on a given client is typically based on the usage of the mobile device by a particular user. Hence, any particular user's local dataset will not be representative of the population distribution.

   This non-IID property makes federated learning to a hard problem in distributed machine learning. Since data are not shared across devices, none of the devices know the global data distribution. In other words, every device is training a model overfitting to itself. The goal of FL is aggregating those overfitting models to a model that correctly estimates the global data distribution.

2. **Heterogeneous communication cost:** The communication time and cost vary largely because of the location of the device, the medium of the communication (3G, 4G, 5G, Wi-Fi). This heterogeneity is usually ignored in most FL works. However, as the system scales up, the communication overhead will increase, and this variant becomes a crucial factor.

   For this wide coverage, the proposed FL system should leverage the local cluster to reduce the average round-trip time (RTT). This local cluster can perform cheap and fast communication inside and achieve higher communication efficiency. In network science, the addition of the local server plays a role in creating a local cluster and reduce the within-cluster distance.

3. **Heterogeneous computation ability:** This project targets training everywhere. This vision includes various devices in the world. Those diversities include

architecture (CPU, memory size) and device status (power, workload)[2]. Classic FL works trained a single model fitting into all the devices. Thus the device which does not match the model requirement will be excluded in this classic FL system. However, data in each device are unique. Despite less contribution, those rare weak devices should be considered in the FL scenario.

3.3 Iteration-level training strategy

To exploit the hierarchical network and alleviate the effect from heterogeneous communication costs, we increase the frequency of local communication (from compute nodes to local servers) and make global communication (from local servers to the global server) rare.

We formally configure our communication system by hyperparameters E_1, E_2. Shown in algorithm 1 and 2, the local servers broadcast the model into their corresponding leaf nodes (line 3 to 5). After both local servers (line 7 to 9 and local nodes (line 2 to 4) train for E_1 epochs, the local servers aggregate the updated models from local nodes (line 10 to 12) and continue next round. The local average model will be uploaded to the global server. We illustrate an example with the setting $(E_1, E_2) = (3, 6)$ in fig.2, in which $E_1 = 3$ infers the number of communication for a compute node required to send the update to the local server, and $E_2 = 6$ indicates the same number, but to the global server. Since compute nodes need the local server to send the update to the global server, E_2 has to be a multiple of E_1, and $E_2 > E_1$.

Algorithm 1 Hierarchical training (on local server)

Input: Local data $\mathbf{X}$, y, global parameter θ
Output: None (The updated parameter θ will be sent to the global server)

1: **while** Ping by global server **do**
2: **for** $i = 1, 2, \ldots, E_2/E_1$ **do**
3: **for** $j = \{1, 2, \ldots, N_{client}\}$ **do**
4: θ_j = Send(node$_j$, θ)
5: **end for**
6: $\theta_0 \leftarrow \theta$
7: **for** $j = \{1, 2, \ldots, E_1\}$ **do**
8: θ_0 =LocalTrain($\mathbf{X}$, y, θ_0)
9: **end for**
10: **for** $j = \{1, 2, \ldots, N_{client}\}$ **do**
11: θ_j = Recv(j)
12: **end for**
13: $\theta \leftarrow \frac{1}{N_{client}+1}(\theta_0 + \sum_{j=1}^{N_{client}} \theta_j)$
14: **end for**
15: Send(θ, Global_server)
16: **end while**

Algorithm 2 Hierarchical training (on local node)

Input: Local data $\mathbf{X}$, y, global parameter θ
Output: None (The updated parameter θ will be sent to the local server)

1: **while** ping by Local server **do**
2: **for** $i = \{1, 2, \ldots, E_1\}$ **do**
3: θ =LocalTrain($\mathbf{X}$, y, θ)
4: **end for**
5: Send(θ, Local_server)
6: **end while**

3.3.1 Baseline training strategy

Prior FL methods like [6] and [9] do not consider the node network in their application. We can reproduce their settings by assigning $E_2 = E_1$. That is, when the compute node sends the updated model to the local server, it also sends it to the global server. This is the baseline without any architectural assumption, and the communication cost is expensive due to the frequent global communication.

3.3.2 For heterogeneous communication

From our assumption in section 3.2, we want to prevent high communication cost from server to server, but require different nodes to share information between them. Thus, we increase E_1 to have more local sharing inside the coverage of a local server and decrease E_2 to reduce the communication overhead in global sharing.

In terms of information gain, discriminative E_1 and E_2 make a trade-off between local sharing and global sharing. When ratio E_1/E_2 is high, compute nodes get information from their neighbor nodes.

In terms of accuracy, since this hierarchy cannot get benefit in communication round number, which is the most common metric in FL, this imbalanced aggregation affects the performance is not widely discussed in prior works. This is one of the focus in this work: How tolerant federated learning is for the imbalanced neighborhood.

3.4 Model-level training strategy

Training on distributed systems is suffered from non-IID data and significant communication overhead. In this project, we target on communication efficiency of the FL. This work focus on the practical implementation, in which all the operation, including communication, computation, and value update, will not be the fixed number. This uncertainty is very crucial in a practical FL system but seldom be mentioned in most of the works.

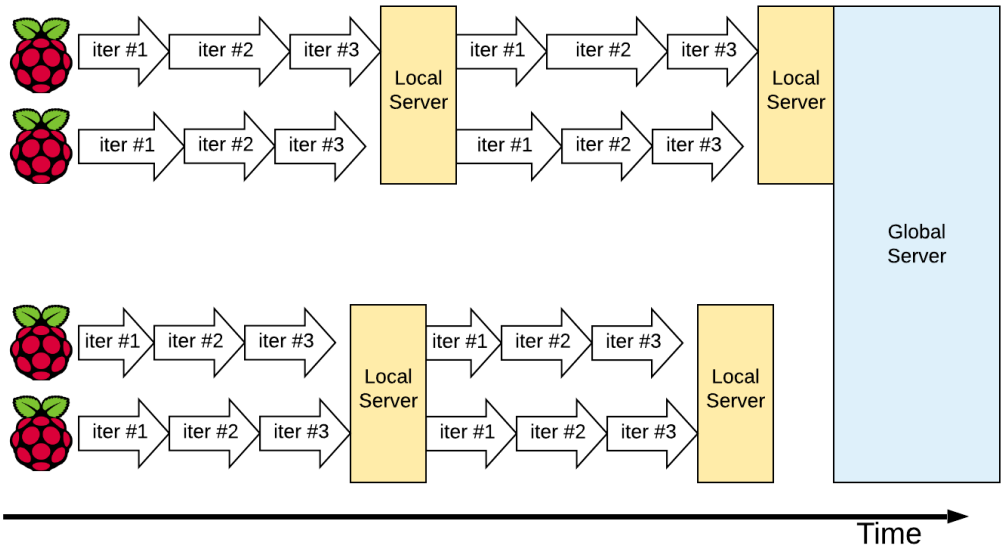

Figure 2. Iteration-level training strategy. The models from each device merge on the local server after three local training rounds. Moreover, the global model forms by averaging the models from local servers.

4. Experimental setup and results

4.1 Platform implementation

In FL, the model parameters are transmitted between the server and nodes. To communicate between the server and nodes, a PyTorch model is parsed into JSON format, which is an open-standard file format that uses human-readable text for the transmission, and the JSON file is sent/received from Transmission Control Protocol (TCP). Each node acts as a TCP server, waiting for a service request in the network. When the global model is available, that model will be sent to each node. This model transmission is a request from the central parameter server to each node.[2]

The communication contains two parts in FL: Broadcasting the global model and aggregating the locally trained model. Ideally, the broadcasting is performed at the beginning of each round, and the parameter server asynchronously aggregates from each node. In python, the TCP package is default point-to-point communication. To implement this broadcast-aggregation pattern, we do the communication sequentially. Accurately, the server will send the model to node one first, then waiting for the weight update from that node. This send/receive will be repeated for all nodes. After that, the parameter server will do the aggregation and obtain the new global model. This is the baseline of the system. In the following milestones, we will optimize this flow and achieve better performance.

[2]Although the parameter server, responsible for aggregating the parameters and broadcasting the updated model, it controls all the parameter update. This parameter server acts as a client under TCP infrastructure; in the opposite, a node passively receiving the updated model is a TCP server. These confusing roles match their characteristic in TCP: An active client sends the request, and a passive server handles that.

4.2 Communication time measurement

To demonstrate the heterogeneity between medium, the nodes in hierarchical FL system communicate in two different ways: The local communications are performed under the Ethernet network from a router; in contrast, the global communications deploys the public Wi-Fi.

	Medium	Mean	Stdev
Local communication	Ethernet	13.2	1.7
Global communicaion	On-campus Wi-Fi	160	86.7

Table 1. Communication measurement on the real system

Shown in table 1, the local/global update takes 13.2/160 ms in average. This 10 times difference reflects the intuition of this work.

In spite of 10-times difference in cost, those communications are still on-campus and very fast. We conduct a small

experiment to ping servers in the world. Although ping operations do not consider the large packet size, the huge gap between ping time demonstrates the potential and importance to solve the problems in heterogeneous communication cost.

Domain	Wired RPi	Wireless RPi	Google	Baidu
Ping time	0.8 ms	6 ms	13.4 ms	1600 ms

Table 2. Ping time to different servers

Shown in table 2, on-campus wireless communication is two times faster than off-campus wired. In other words, the local fast communication in our work can get more benefit if we apply our system to the real world.

4.3 Algorithm implementation

To evaluate the effectiveness of our method, we use the Raspberry Pi to reproduce the real scenario. The system consists of 8 Raspberry Pis as a physical client, a laptop as the central parameter server. The Raspberry Pis connect to the server through a wired network.

A Convolutional Neural Network (CNN) model consisting of two convolutional layers and two fully-connected layers is used for experiments in this work. We train this CNN with 7 thousand parameters on the non-IID MNIST dataset[3], and each device contains only a part of the entire dataset. At each communication round, the Raspberry Pi and all processes are involved in the training. The training process lasts 100 communication rounds; the mini-batch size of the model in Raspberry Pi is 16, but the mini-batch size in other processes is 100. The learning rate is initialized to 0.1 and decays by 0.996.

The test accuracy of FedAvg and FedMAX are shown in Fig.3 As we can see, the accuracy can get about 77%, which means the two-layer CNN is enough for the MNIST dataset. Furthermore, it shows that the Raspberry Pi can successfully communicate with the server and perform the training locally. However, the test accuracy of FedAvg and FedMAX is similar, which means FedMAX cannot provide better communication efficiency in this case. We assume the FedMAX can provide a regularization of the parameters of CNN. However, with only about 7,000 parameters, it is hard to see the improvement compared with a larger neural network with about millions of parameters.

The test accuracy of FedAvg with different network structure is shown in Fig.4. The blue line represents the accuracy of the star network, and the red line represents the accuracy of the hierarchical network, with configuration $E_1 = 6$ and $E_2 = 3$. In this figure, we observe the higher accuracy at the first 500 seconds in communication. This demonstrates the hierarchical network has a faster convergence speed than the network without a hierarchical structure.

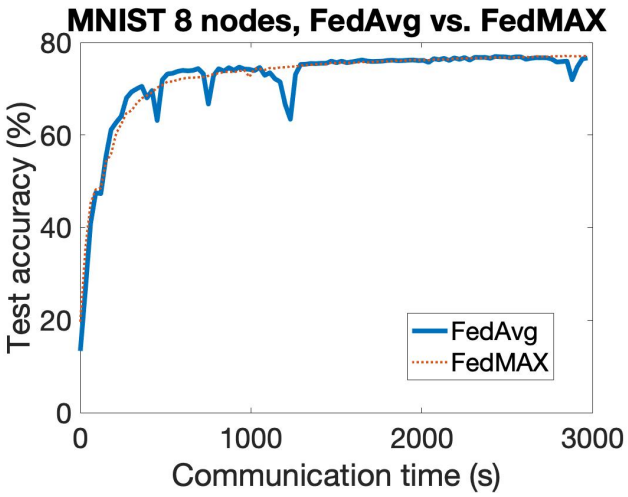

Figure 3. Test accuracy of FedAvg and FedMAX from the network of edge devices. The network contains 8 nodes as clients and 1 node as central server. Each client node represents a Raspberry Pi and each server node represents a laptop.

Another noting point is the time achieve 75% accuracy. Although with faster training speed at the beginning, the hierarchical network has a longer time to reach 75% accuracy. The possible explanation might be the lack of global communication. The star network takes longer time per epoch to aggregate all the model in different nodes. This implies a more intense knowledge exchange between local models. However, in our hierarchical network, the training is still restricted by the less global communication during the latter stages. We might apply a more sophisticated strategy to make a tradeoff between accuracy and speed along the entire training process, but this is beyond our scope in this work.

4.4 Algorithm exploration

Apart from the algorithm implementation, multiple approaches are explored to deal with the device heterogeneity. In this setting, we simulate a total of 100 local devices, and half of them are designed as poor devices, which take longer time to perform local training and have limited storage space; half of them are designed as good devices, which can perform fast local training and have more space for the model.

During each communication round, 10% of these devices are randomly selected for local training. The training process lasts 600 communication rounds; the mini-batch size at

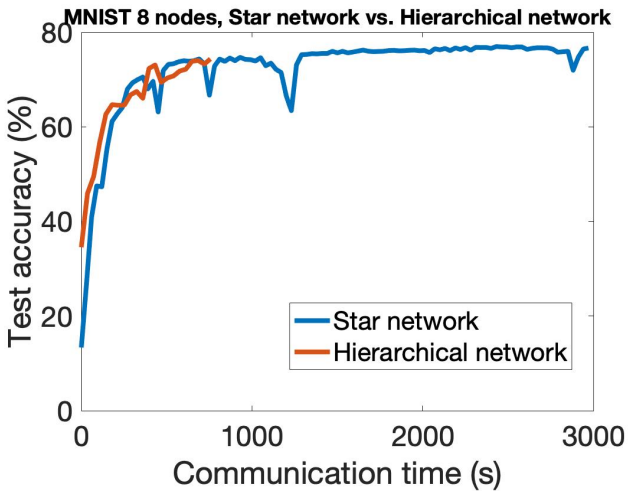

Figure 4. Test accuracy of FedAvg with different network structure. The blue line represents star network, and the red line represents hierarchical network.

each selected device is 100. The learning rate is initialized to 0.1 and decays by 0.996.

In order to make the poor devices have similar training speed as good devices, different models are assigned to different kinds of devices. The poor devices are trained with reduced CNN, which has fewer convolutional kernels at each layer; instead, good devices keep training with normal CNN. On the server, the different models are aggregated by merging the small model with a random but fix the position of the larger model. The approach can finally get 37.65% accuracy.

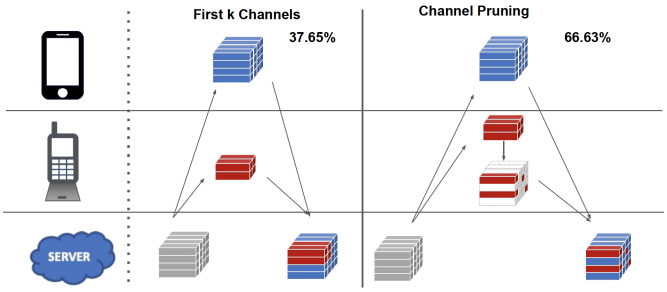

Figure 5. Using different models for powerful and poor devices: (a) Merge the small model with first k channels of larger model, (b) Prune the large model into smaller one, after training map the small model back to the large model, and merge large model and mapped large model

Approach	First k channels	Channel pruning
Test accuracy	37.65%	66.63%

Table 3. Final accuracy of different merging approach

Another approach to deal with the device heterogeneity is using the model pruning[4] to form a small model and record the index of pruned channels, as shown in Fig.5. After the local training, the small model maps back to the shape of the original model w.r.t the index recorded in the dictionary. This mapped model can be directly averaged with the large model to form a new global model. This approach can finally get 66.63% accuracy.

4.5 Illustration

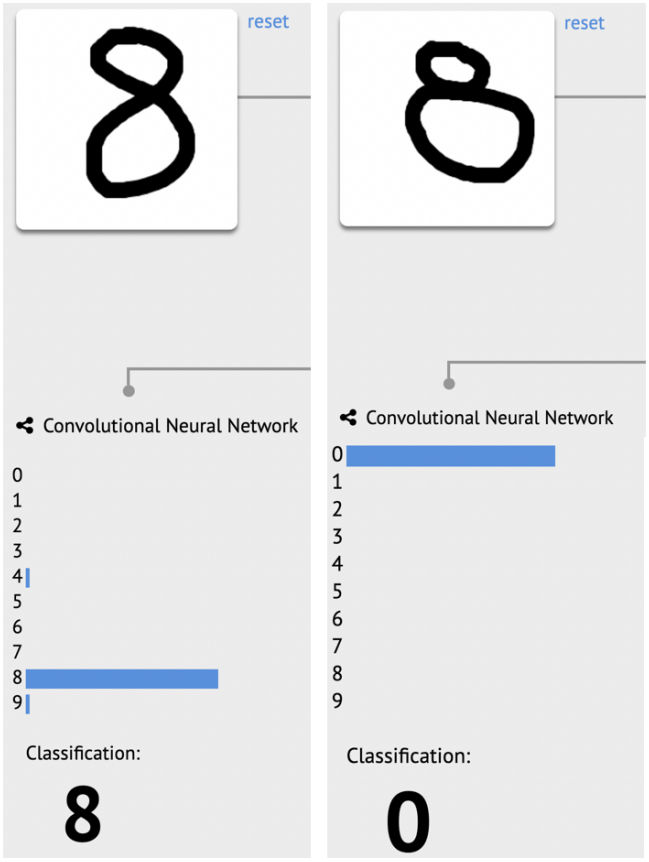

Figure 6. Digit classification illustration. The neural network can give a right classification or wrong classification

After we train the model in eight Raspberry Pi devices for 100 rounds, our model can finally get 75% accuracy.

We use an interactive demo to show the results of model, as shown in Fig.6. From the left figure we can see the "8" is correctly classified, but on the right side, "8" is misclassified. One reason is our model can only get 75% accuracy which is not high enough, so that the model cannot perform the right classification sometimes. Another reason is even with high accuracy on MNIST test dataset, the model still has chance to classify the digit wrongly. The digit written on the board may have different characteristic from the MNIST dataset, so that the model can be easily fooled by the plot. Therefore, it is important to find a way to have a model that can generalize[8] well to different distribution.

5. Conclusion

In this work, we implemented the communication-efficient algorithm FedAvg and FedMAX on the hardware devices and measured the training time and test accuracy of these two algorithms. The test accuracy of FedAvg and FedMAX are similar due to the small size of the training model.

We also addressed the hardware heterogeneity by assigning different sizes of models for devices with different abilities by pruning the model. We verified that the algorithm can be implemented in hierarchical structure network, which means our work can be possibly scaled to larger setting with more than 8 devices. We plan to increase the $E_1/E_2 = 2$ to save more communication time.

In the future, We plan to train a larger model to make the most of FedMAX. We will measure the performance of devices with different algorithms and communication latency during the training process.

References

[1] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[2] R. Buchty, V. Heuveline, W. Karl, and J.-P. Weiss. A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators. *Concurrency and Computation: Practice and Experience*, 24(7):663–675, 2012.

[3] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[4] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[5] B. Kartikeya, C. Wei, and M. Radu. Fedmax: Activation entropy maximization targeting effective non-iid federated learning. 2019.

[6] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[7] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[8] M. Vidyasagar. *A theory of learning and generalization*. Springer-Verlag, 2002.

[9] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.