Bayesian Nonparametric Federated Learning of Neural Networks

Mikhail Yurochkin [1 2] **Mayank Agarwal** [1 2] **Soumya Ghosh** [1 2 3] **Kristjan Greenewald** [1 2] **Trong Nghia Hoang** [1 2]
Yasaman Khazaeni [1 2]

Abstract

In federated learning problems, data is scattered across different servers and exchanging or pooling it is often impractical or prohibited. We develop a Bayesian nonparametric framework for federated learning with neural networks. Each data server is assumed to provide local neural network weights, which are modeled through our framework. We then develop an inference approach that allows us to synthesize a more expressive global network without additional supervision, data pooling and with as few as a single communication round. We then demonstrate the efficacy of our approach on federated learning problems simulated from two popular image classification datasets.[1]

1. Introduction

The standard machine learning paradigm involves algorithms that learn from centralized data, possibly pooled together from multiple data sources. The computations involved may be done on a single machine or farmed out to a cluster of machines. However, in the real world, data often live in silos and amalgamating them may be prohibitively expensive due to communication costs, time sensitivity, or privacy concerns. Consider, for instance, data recorded from sensors embedded in wearable devices. Such data is inherently private, can be voluminous depending on the sampling rate of the sensors, and may be time sensitive depending on the analysis of interest. Pooling data from many users is technically challenging owing to the severe computational burden of moving large amounts of data, and is fraught with privacy concerns stemming from potential data breaches that may expose a user's protected health information (PHI).

Federated learning addresses these pitfalls by obviating the need for centralized data, instead designing algorithms that learn from sequestered data sources. These algorithms iterate between training local models on each data source and distilling them into a global federated model, all without explicitly combining data from different sources. Typical federated learning algorithms, however, require access to locally stored data for learning. A more extreme case surfaces when one has access to models pre-trained on local data but not the data itself. Such situations may arise from catastrophic data loss but increasingly also from regulations such as the general data protection regulation (GDPR) (EU, 2016), which place severe restrictions on the storage and sharing of personal data. Learned models that capture only aggregate statistics of the data can typically be disseminated with fewer limitations. A natural question then is, can "legacy" models trained independently on data from different sources be combined into an improved federated model?

Here, we develop and carefully investigate a probabilistic federated learning framework with a particular emphasis on training and aggregating neural network models. We assume that either local data or pre-trained models trained on local data are available. When data is available, we proceed by training local models for each data source, *in parallel*. We then match the estimated local model parameters (groups of weight vectors in the case of neural networks) across data sources to construct a global network. The matching, to be formally defined later, is governed by the posterior of a Beta-Bernoulli process (BBP) (Thibaux & Jordan, 2007), a Bayesian nonparametric (BNP) model that allows the local parameters to either match existing global ones or to create new global parameters if existing ones are poor matches.

Our construction provides several advantages over existing approaches. First, it decouples the learning of local models from their amalgamation into a global federated model. This decoupling allows us to remain agnostic about the local learning algorithms, which may be adapted as necessary, with each data source potentially even using a different learning algorithm. Moreover, given only pre-trained models, our BBP informed matching procedure is able to combine them into a federated global model without requiring additional data or knowledge of the learning algorithms used to generate the pre-trained models. This is

[1]IBM Research, Cambridge [2]MIT-IBM Watson AI Lab [3]Center for Computational Health. Correspondence to: Mikhail Yurochkin <mikhail.yurochkin@ibm.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

[1]Code is available at `https://github.com/IBM/probabilistic-federated-neural-matching`

in sharp contrast with existing work on federated learning of neural networks (McMahan et al., 2017), which require strong assumptions about the local learners, for instance, that they share the same random initialization, and are not applicable for combining pre-trained models. Next, the BNP nature of our model ensures that we recover compressed global models with fewer parameters than the cardinality of the set of all local parameters. Unlike naive ensembles of local models, this allows us to store fewer parameters and perform more efficient inference at test time, requiring only a single forward pass through the compressed model as opposed to J forward passes, once for each local model. While techniques such as knowledge distillation (Hinton et al., 2015) allow for the cost of multiple forward passes to be amortized, training the distilled model itself requires access to data pooled across all sources or an auxiliary dataset, luxuries unavailable in our scenario. Finally, even in the traditional federated learning scenario, where local and global models are learned together, we show empirically that our proposed method outperforms existing distributed training and federated learning algorithms (Dean et al., 2012; McMahan et al., 2017) while requiring far fewer communications between the local data sources and the global model server.

The remainder of the paper is organized as follows. We briefly introduce the Beta-Bernoulli process in Section 2 before describing our model for federated learning in Section 3. We thoroughly evaluate the proposed model and demonstrate its utility empirically in Section 4. Finally, Section 5 discusses current limitations of our work and open questions.

2. Background and Related Works

Our approach builds on tools from Bayesian nonparametrics, in particular the Beta-Bernoulli Process (BBP) (Thibaux & Jordan, 2007) and the closely related Indian Buffet Process (IBP) (Griffiths & Ghahramani, 2011). We briefly review these ideas before describing our approach.

2.1. Beta-Bernoulli Process (BBP)

Let Q be a random measure distributed by a Beta process with mass parameter γ_0 and base measure H. That is, $Q|\gamma_0, H \sim \text{BP}(1, \gamma_0 H)$. It follows that Q is a discrete (*not* probability) measure $Q = \sum_i \mathrm{q}_i \delta_{\theta_i}$ formed by an infinitely countable set of (weight, atom) pairs $(\mathrm{q}_i, \theta_i) \in [0, 1] \times \Omega$. The weights $\{\mathrm{q}_i\}_{i=1}^{\infty}$ are distributed by a stick-breaking process (Teh et al., 2007): $\mathrm{c}_i \sim \text{Beta}(\gamma_0, 1)$, $\mathrm{q}_i = \prod_{j=1}^{i} \mathrm{c}_j$ and the atoms are drawn i.i.d from the normalized base measure $\theta_i \sim H/H(\Omega)$ with domain Ω. In this paper, Ω is simply $\mathbb{R}^D$ for some D. Subsets of atoms in the random measure Q are then selected using a Bernoulli process with a base measure Q. That is, each subset $\mathcal{T}_j$ with $j = 1, \ldots, J$ is characterized by a Bernoulli process with base measure Q, $\mathcal{T}_j|Q \sim \text{BeP}(Q)$. Each subset $\mathcal{T}_j$ is also a discrete measure formed by pairs $(\mathrm{b}_{ji}, \theta_i) \in \{0, 1\} \times \Omega$, $\mathcal{T}_j := \sum_i \mathrm{b}_{ji} \delta_{\theta_i}$, where $\mathrm{b}_{ji}|\mathrm{q}_i \sim \text{Bernoulli}(\mathrm{q}_i)\, \forall i$ is a binary random variable indicating whether atom θ_i belongs to subset $\mathcal{T}_j$. The collection of such subsets is then said to be distributed by a Beta-Bernoulli process.

2.2. Indian Buffet Process (IBP)

The above subsets are conditionally independent given Q. Thus, marginalizing Q will induce dependencies among them. In particular, we have $\mathcal{T}_J|\mathcal{T}_1, \ldots, \mathcal{T}_{J-1} \sim \text{BeP}\left(H\frac{\gamma_0}{J} + \sum_i \frac{m_i}{J}\delta_{\theta_i}\right)$, where $m_i = \sum_{j=1}^{J-1} \mathrm{b}_{ji}$ (dependency on J is suppressed in the notation for simplicity) and is sometimes called the Indian Buffet Process. The IBP can be equivalently described by the following culinary metaphor. Imagine J customers arrive sequentially at a buffet and choose dishes to sample as follows, the first customer tries Poisson(γ_0) dishes. Every subsequent j-th customer tries each of the previously selected dishes according to their popularity, i.e. dish i with probability m_i/j, and then tries Poisson(γ_0/j) new dishes.

The IBP, which specifies a distribution over sparse binary matrices with infinitely many columns, was originally demonstrated for latent factor analysis (Ghahramani & Griffiths, 2005). Several extensions to the IBP (and the equivalent BBP) have been developed, see Griffiths & Ghahramani (2011) for a review. Our work is related to a recent application of these ideas to distributed topic modeling (Yurochkin et al., 2018), where the authors use the BBP for modeling topics learned from multiple collections of document, and provide an inference scheme based on the Hungarian algorithm (Kuhn, 1955).

2.3. Federated and Distributed Learning

Federated learning has garnered interest from the machine learning community of late. Smith et al. (2017) pose federated learning as a multi-task learning problem, which exploits the convexity and decomposability of the cost function of the underlying support vector machine (SVM) model for distributed learning. This approach however does not extend to the neural network structure considered in our work. McMahan et al. (2017) use strategies based on simple averaging of the local learner weights to learn the federated model. However, as pointed out by the authors, such naive averaging of model parameters can be disastrous for non-convex cost functions. To cope, they have to use a scheme where the local learners are forced to share the same random initialization. In contrast, our proposed framework is naturally immune to such issues since its development assumes nothing specific about how the local models were trained. Moreover, unlike existing work in this area, our framework is non-parametric in nature allowing the federated model

to flexibly grow or shrink its complexity (i.e., its size) to account for varying data complexity.

There is also significant work on distributed deep learning (Lian et al., 2015; 2017; Moritz et al., 2015; Li et al., 2014; Dean et al., 2012). However, the emphasis of these works is on scalable training from large data and they typically require frequent communication between the distributed nodes to be effective. Yet others explore distributed optimization with a specific emphasis on communication efficiency (Zhang et al., 2013; Shamir et al., 2014; Yang, 2013; Ma et al., 2015; Zhang & Lin, 2015). However, as pointed out by McMahan et al. (2017), these works primarily focus on settings with convex cost functions and often assume that each distributed data source contains an equal number of data instances. These assumptions, in general, do not hold in our scenario. Finally, neither these distributed learning approaches nor existing federated learning approaches decouple local training from global model aggregation. As a result they are not suitable for combining pre-trained legacy models, a particular problem of interest in this paper.

3. Probabilistic Federated Neural Matching

We now describe how the Bayesian nonparametric machinery can be applied to the problem of federated learning with neural networks. Our goal will be to identify subsets of neurons in each of the J local models that match neurons in other local models. We will then appropriately combine the matched neurons to form a global model.

Our approach to federated learning builds upon the following basic problem. Suppose we have trained J Multilayer Perceptrons (MLPs) with one hidden layer each. For the jth MLP $j = 1, \ldots, J$, let $V_j^{(0)} \in \mathbb{R}^{D \times L_j}$ and $\tilde{v}_j^{(0)} \in \mathbb{R}^{L_j}$ be the weights and biases of the hidden layer; $V_j^{(1)} \in \mathbb{R}^{L_j \times K}$ and $\tilde{v}_j^{(1)} \in \mathbb{R}^K$ be weights and biases of the softmax layer; D be the data dimension, L_j the number of neurons on the hidden layer; and K the number of classes. We consider a simple architecture: $f_j(x) = \text{softmax}(\sigma(xV_j^{(0)} + \tilde{v}_j^{(0)})V_j^{(1)} + \tilde{v}_j^{(1)})$ where $\sigma(\cdot)$ is some nonlinearity (sigmoid, ReLU, etc.). Given the collection of weights and biases $\{V_j^{(0)}, \tilde{v}_j^{(0)}, V_j^{(1)}, \tilde{v}_j^{(1)}\}_{j=1}^J$ we want to learn a global neural network with weights and biases $\Theta^{(0)} \in \mathbb{R}^{D \times L}, \tilde{\theta}^{(0)} \in \mathbb{R}^L, \Theta^{(1)} \in \mathbb{R}^{L \times K}, \tilde{\theta}^{(1)} \in \mathbb{R}^K$, where $L \ll \sum_{j=1}^J L_j$ is an unknown number of hidden units of the global network to be inferred.

Our first observation is that ordering of neurons of the hidden layer of an MLP is permutation invariant. Consider any permutation $\tau(1, \ldots, L_j)$ of the j-th MLP – reordering columns of $V_j^{(0)}$, biases $\tilde{v}_j^{(0)}$ and rows of $V_j^{(1)}$ according to $\tau(1, \ldots, L_j)$ will not affect the outputs $f_j(x)$ for any value of x. Therefore, instead of treating weights as matrices and biases as vectors we view them as unordered collections of vectors $V_j^{(0)} = \{v_{jl}^{(0)} \in \mathbb{R}^D\}_{l=1}^{L_j}$, $V_j^{(1)} = \{v_{jl}^{(1)} \in \mathbb{R}^{L_j}\}_{l=1}^K$ and scalars $\tilde{v}_j^{(0)} = \{\tilde{v}_{jl}^{(0)} \in \mathbb{R}\}_{l=1}^{L_j}$ correspondingly.

Hidden layers in neural networks are commonly viewed as feature extractors. This perspective can be justified by the fact that the last layer of a neural network classifier simply performs a softmax regression. Since neural networks often outperform basic softmax regression, they must be learning high quality feature representations of the raw input data. Mathematically, in our setup, every hidden neuron of the j-th MLP represents a new feature $\tilde{x}_l(v_{jl}^{(0)}, \tilde{v}_{jl}^{(0)}) = \sigma(\langle x, v_{jl}^{(0)} \rangle + \tilde{v}_{jl}^{(0)})$. Our second observation is that each $(v_{jl}^{(0)}, \tilde{v}_{jl}^{(0)})$ parameterizes the corresponding neuron's feature extractor. Since, the J MLPs are trained on the same general type of data (not necessarily homogeneous), we assume that they share at least some feature extractors that serve the same purpose. However, due to the permutation invariance issue discussed previously, a feature extractor indexed by l from the j-th MLP is unlikely to correspond to a feature extractor with the same index from a different MLP. In order to construct a set of global feature extractors (neurons) $\{\theta_i^{(0)} \in \mathbb{R}^D, \tilde{\theta}_i^{(0)} \in \mathbb{R}\}_{i=1}^L$ we must model the process of grouping and combining feature extractors of collection of MLPs.

3.1. Single Layer Neural Matching

We now present the key building block of our framework, a Beta Bernoulli Process (Thibaux & Jordan, 2007) based model of MLP weight parameters. Our model assumes the following generative process. First, draw a collection of global atoms (hidden layer neurons) from a Beta process prior with a base measure H and mass parameter γ_0, $Q = \sum_i \mathrm{q}_i \delta_{\theta_i}$. In our experiments we choose $H = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ as the base measure with $\boldsymbol{\mu}_0 \in \mathbb{R}^{D+1+K}$ and diagonal Σ_0. Each $\theta_i \in \mathbb{R}^{D+1+K}$ is a concatenated vector of $[\theta_i^{(0)} \in \mathbb{R}^D, \tilde{\theta}_i^{(0)} \in \mathbb{R}, \theta_i^{(1)} \in \mathbb{R}^K]$ formed from the feature extractor weight-bias pairs with the corresponding weights of the softmax regression. In what follows, we will use "batch" to refer to a partition of the data.

Next, for each $j = 1, \ldots, J$ select a subset of the global atoms for batch j via the Bernoulli process:

$$\mathcal{T}_j := \sum_i \mathrm{b}_{ji} \delta_{\theta_i}, \text{ where } \mathrm{b}_{ji} | \mathrm{q}_i \sim \text{Bern}(\mathrm{q}_i) \, \forall i. \quad (1)$$

$\mathcal{T}_j$ is supported by atoms $\{\theta_i : \mathrm{b}_{ji} = 1, i = 1, 2, \ldots\}$, which represent the identities of the atoms (neurons) used by batch j. Finally, assume that observed local atoms are noisy measurements of the corresponding global atoms:

$$\mathbf{v}_{jl} | \mathcal{T}_j \sim \mathcal{N}(\mathcal{T}_{jl}, \boldsymbol{\Sigma}_j) \text{ for } l = 1, \ldots, L_j; \;\; L_j := \text{card}(\mathcal{T}_j), \quad (2)$$

with $\mathbf{v}_{jl} = [v_{jl}^{(0)}, \tilde{v}_{jl}^{(0)}, v_{jl}^{(1)}]$ being the weights, biases, and softmax regression weights corresponding to the l-th neuron of the j-th MLP trained with L_j neurons on the data of batch j.

Under this model, the key quantity to be inferred is the collection of random variables that **match** observed atoms (neurons) at any batch to the global atoms. We denote the collection of these random variables as $\{\boldsymbol{B}^j\}_{j=1}^J$, where $\boldsymbol{B}_{i,l}^j = 1$ implies that $\mathcal{T}_{jl} = \theta_i$ (there is a one-to-one correspondence between $\{\mathrm{b}_{ji}\}_{i=1}^\infty$ and $\boldsymbol{B}^j$).

Maximum a posteriori estimation. We now derive an algorithm for MAP estimation of global atoms for the model presented above. The objective function to be maximized is the posterior of $\{\theta_i\}_{i=1}^\infty$ and $\{\boldsymbol{B}^j\}_{j=1}^J$:

$$\begin{aligned}\arg\max_{\{\theta_i\},\{\boldsymbol{B}^j\}} & P(\{\theta_i\},\{\boldsymbol{B}^j\}|\{\mathbf{v}_{jl}\}) \\ & \propto P(\{\mathbf{v}_{jl}\}|\{\theta_i\},\{\boldsymbol{B}^j\})P(\{\boldsymbol{B}^j\})P(\{\theta_i\}).\end{aligned} \tag{3}$$

Note that the next proposition easily follows from Gaussian-Gaussian conjugacy:

Proposition 1. *Given $\{\boldsymbol{B}^j\}$, the MAP estimate of $\{\theta_i\}$ is given by*

$$\hat{\theta}_i = \frac{\boldsymbol{\mu}_0/\sigma_0^2 + \sum_{j,l} B_{i,l}^j \mathbf{v}_{jl}/\sigma_j^2}{1/\sigma_0^2 + \sum_{j,l} B_{i,l}^j/\sigma_j^2} \text{ for } i = 1,\dots,L, \tag{4}$$

where for simplicity we assume $\boldsymbol{\Sigma}_0 = \boldsymbol{I}\sigma_0^2$ and $\boldsymbol{\Sigma}_j = \boldsymbol{I}\sigma_j^2$.

Using this fact we can cast optimization corresponding to (3) with respect to only $\{\boldsymbol{B}^j\}_{j=1}^J$. Taking the natural logarithm we obtain:

$$\arg\max_{\{\boldsymbol{B}^j\}} \frac{1}{2}\sum_i \frac{\left\|\frac{\boldsymbol{\mu}_0}{\sigma_0^2} + \sum_{j,l} B_{i,l}^j \frac{\mathbf{v}_{jl}}{\sigma_j}^2\right\|^2}{1/\sigma_0^2 + \sum_{j,l} B_{i,l}^j/\sigma_j^2} + \log(P(\{\boldsymbol{B}^j\}). \tag{5}$$

We consider an iterative optimization approach: fixing all but one $\boldsymbol{B}^j$ we find corresponding optimal assignment, then pick a new j at random and proceed until convergence. In the following we will use notation $-j$ to denote "all but j". Let $L_{-j} = \max\{i : B_{i,l}^{-j} = 1\}$ denote number of active global weights outside of group j. We now rearrange the *first* term of (5) by partitioning it into $i = 1,\dots,L_{-j}$ and $i = L_{-j}+1,\dots,L_{-j}+L_j$. We are interested in solving for $\boldsymbol{B}^j$, hence we can modify the objective function by subtracting terms independent of $\boldsymbol{B}^j$ and noting that $\sum_l B_{i,l}^j \in \{0,1\}$, i.e. it is 1 if some neuron from batch j is matched to global neuron i and 0 otherwise:

$$\begin{aligned}&\frac{1}{2}\sum_i \frac{\|\boldsymbol{\mu}_0/\sigma_0^2 + \sum_{j,l} B_{i,l}^j \mathbf{v}_{jl}/\sigma_j^2\|^2}{1/\sigma_0^2 + \sum_{j,l} B_{i,l}^j/\sigma_j^2} = \\ &\sum_{i=1}^{L_{-j}+L_j}\sum_{l=1}^{L_j} B_{i,l}^j \left(\frac{\|\boldsymbol{\mu}_0/\sigma_0^2 + \mathbf{v}_{jl}/\sigma_j^2 + \sum_{-j,l} B_{i,l}^j \mathbf{v}_{jl}/\sigma_j^2\|^2}{1/\sigma_0^2 + 1/\sigma_j^2 + \sum_{-j,l} B_{i,l}^j/\sigma_j^2}\right. \\ &\qquad \left. - \frac{\|\boldsymbol{\mu}_0/\sigma_0^2 + \sum_{-j,l} B_{i,l}^j \mathbf{v}_{jl}/\sigma_j^2\|^2}{1/\sigma_0^2 + \sum_{-j,l} B_{i,l}^j/\sigma_j^2}\right).\end{aligned} \tag{6}$$

Now we consider the *second* term of (5):

$$\log P(\{\boldsymbol{B}^j\}) = \log P(\boldsymbol{B}^j|\boldsymbol{B}^{-j}) + \log P(\boldsymbol{B}^{-j}).$$

First, because we are optimizing for $\boldsymbol{B}^j$, we can ignore $\log P(\boldsymbol{B}^{-j})$. Second, due to exchangeability of batches (i.e. customers of the IBP), we can always consider $\boldsymbol{B}^j$ to be the last batch (i.e. last customer of the IBP). Let $m_i^{-j} = \sum_{-j,l} B_{i,l}^j$ denote number of times batch weights were assigned to global weight i outside of group j. We then obtain:

$$\begin{aligned}\log P(\{\boldsymbol{B}^j\}) = &\sum_{i=1}^{L_{-j}}\sum_{l=1}^{L_j} B_{i,l}^j \log\frac{m_i^{-j}}{J - m_i^{-j}} \\ &+ \sum_{i=L_{-j}+1}^{L_{-j}+L_j}\sum_{l=1}^{L_j} B_{i,l}^j \left(\log\frac{\gamma_0}{J} - \log(i - L_{-j})\right).\end{aligned} \tag{7}$$

Combining (6) and (7) we obtain the assignment cost objective, which we solve with the Hungarian algorithm.

Proposition 2. *The (negative) assignment cost specification for finding $\boldsymbol{B}^j$ is $-C_{i,l}^j =$*

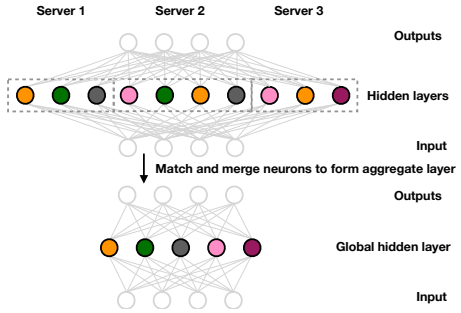
$$\begin{cases} \frac{\left\|\frac{\boldsymbol{\mu}_0}{\sigma_0^2} + \frac{\mathbf{v}_{jl}}{\sigma_j^2} + \sum_{-j,l} B_{i,l}^j \frac{\mathbf{v}_{jl}}{\sigma_j^2}\right\|^2}{\frac{1}{\sigma_0^2} + \frac{1}{\sigma_j^2} + \sum_{-j,l} B_{i,l}^j/\sigma_j^2} - \frac{\left\|\frac{\boldsymbol{\mu}_0}{\sigma_0^2} + \sum_{-j,l} B_{i,l}^j \frac{\mathbf{v}_{jl}}{\sigma_j^2}\right\|^2}{\frac{1}{\sigma_0^2} + \sum_{-j,l} B_{i,l}^j/\sigma_j^2} + 2\log\frac{m_i^{-j}}{J - m_i^{-j}}, \\ \qquad\qquad i \le L_{-j} \\ \frac{\left\|\frac{\boldsymbol{\mu}_0}{\sigma_0^2} + \frac{\mathbf{v}_{jl}}{\sigma_j^2}\right\|^2}{\frac{1}{\sigma_0^2} + \frac{1}{\sigma_j^2}} - \frac{\left\|\frac{\boldsymbol{\mu}_0}{\sigma_0^2}\right\|^2}{\frac{1}{\sigma_0^2}} - 2\log\frac{i - L_{-j}}{\gamma_0/J}, \quad L_{-j} < i \le L_{-j} + L_j. \end{cases} \tag{8}$$

We then apply the Hungarian algorithm to find the minimizer *of $\sum_i \sum_l B_{i,l}^j C_{i,l}^j$ and obtain the neuron matching assignments. Proof is described in Supplement section 1.*

We summarize the overall single layer inference procedure in Figure 1 below.

3.2. Multilayer Neural Matching

The model we have presented thus far can handle any arbitrary width single layer neural network, which is known to be theoretically sufficient for approximating any function of interest (Hornik et al., 1989). However, deep neural



Algorithm 1 Single Layer Neural Matching

1: Collect weights and biases from the J batches and form $\mathbf{v}_{jl}$.
2: Form assignment cost matrix per (8).
3: Compute matching assignments B^j using the Hungarian algorithm.
4: Enumerate all resulting unique global neurons and use (4) to infer the associated global weight vectors from all instances of the global neurons across the J batches.
5: Concatenate the global neurons and the inferred weights and biases to form the new global hidden layer.

Figure 1: Single layer Probabilistic Federated Neural Matching algorithm showing matching of three MLPs. Nodes in the graphs indicate neurons, neurons of the same color have been matched. Our approach consists of using the corresponding neurons in the output layer to convert the neurons in each of the J batches to weight vectors referencing the output layer. These weight vectors are then used to form a cost matrix, which the Hungarian algorithm then uses to do the matching. The matched neurons are then aggregated via Proposition 1 to form the global model.

networks with moderate layer widths are known to be beneficial both practically (LeCun et al., 2015) and theoretically (Poggio et al., 2017). We extend our neural matching approach to these deep architectures by defining a generative model of deep neural network weights from outputs back to inputs (top-down). Let C denote the number of hidden layers and L^c the number of neurons on the c-th layer. Then $L^{C+1} = K$ is the number of labels and $L^0 = D$ is the input dimension. In the top down approach, we consider the global atoms to be vectors of outgoing weights from a neuron instead of weights forming a neuron as it was in the single hidden layer model. This change is needed to avoid base measures with unbounded dimensions.

Starting with the top hidden layer $c = C$, we generate each layer following a model similar to that used in the single layer case. For each layer we generate a collection of global atoms and select a subset of them for each batch using Beta-Bernoulli process construction. L^{c+1} is the number of neurons on the layer $c+1$, which controls the dimension of the atoms in layer c.

Definition 1 (Multilayer generative process)**.** *Starting with layer $c = C$, generate (as in the single layer process)*

$$Q^c|\gamma_0^c, H^c, L^{c+1} \sim \mathrm{BP}(1, \gamma_0^c H^c), \tag{9}$$
$$\text{then } Q^c = \sum_i q_i^c \delta_{\theta_i^c},\ \theta_i^c \sim \mathcal{N}(\boldsymbol{\mu}_0^c, \boldsymbol{\Sigma}_0^c),\ \boldsymbol{\mu}_0^c \in \mathbb{R}^{L^{c+1}}$$
$$\mathcal{T}_j^c := \sum_i \mathrm{b}_{ji}^c \delta_{\theta_i^c},\ \text{where } \mathrm{b}_{ji}^c|\mathrm{q}_i^c \sim \mathrm{Bern}(\mathrm{q}_i^c).$$

This $\mathcal{T}_j^c$ is the set of global atoms (neurons) used by batch j in layer c, it contains atoms $\{\theta_i^c : \mathrm{b}_{ji}^c = 1, i = 1, 2, \ldots\}$. Finally, generate the observed local atoms:

$$\mathbf{v}_{jl}^c|\mathcal{T}_j^c, \sim \mathcal{N}(\mathcal{T}_{jl}^c, \boldsymbol{\Sigma}_j^c) \text{ for } l = 1, \ldots, L_j^c, \tag{10}$$

where we have set $L_j^c := \mathrm{card}(\mathcal{T}_j^c)$. Next, compute the generated number of global neurons $L^c = \mathrm{card}\{\cup_{j=1}^J \mathcal{T}_j^c\}$ and repeat this generative process for the next layer $c-1$. Repeat until all layers are generated ($c = C, \ldots, 1$).

An important difference from the single layer model is that we should now set to 0 some of the dimensions of $\mathbf{v}_{jl}^c \in \mathbb{R}^{L^{c+1}}$ since they correspond to weights outgoing to neurons of the layer $c+1$ not present on the batch j, i.e. $\mathbf{v}_{jli}^c := 0$ if $\mathrm{b}_{ji}^{c+1} = 0$ for $i = 1, \ldots, L^{c+1}$. The resulting model can be understood as follows. There is a global fully connected neural network with L^c neurons on layer c and there are J partially connected neural networks with L_j^c active neurons on layer c, while weights corresponding to the remaining $L^c - L_j^c$ neurons are zeroes and have no effect locally.

Remark 1. *Our model can conceptually handle permuted ordering of the input dimensions across batches, however in most practical cases the ordering of input dimensions is consistent across batches. Thus, we assume that the weights connecting the first hidden layer to the inputs exhibit permutation invariance only on the side of the first hidden layer. Similarly to how all weights were concatenated in the single hidden layer model, we consider $\boldsymbol{\mu}_0^c \in \mathbb{R}^{D+L^{c+1}}$ for $c = 1$. We also note that the bias term can be added to the model, we omitted it to simplify notation.*

Inference Following the top-down generative model, we adopt a greedy inference procedure that first infers the matching of the top layer and then proceeds down the layers of the network. This is possible because the generative process for each layer depends only on the identity and number of the global neurons in the layer above it, hence once we infer the $c+1$th layer of the global model we can apply the single layer inference algorithm (Algorithm 1) to the cth layer. This greedy setup is illustrated in Figure 1 in Supplement section 2. The per-layer inference follows directly from the single layer case, yielding the following propositions.

Proposition 3. *The (negative) assignment cost specification for finding $\boldsymbol{B}^{j,c}$ is $-C_{i,l}^{j,c} =$*

$$\begin{cases} \frac{\left\| \frac{\boldsymbol{\mu}_0^c}{(\sigma_0^c)^2} + \frac{\mathbf{v}_{jl}^c}{(\sigma_j^c)^2} + \sum_{-j,l} B_{i,l}^{j,c} \frac{\mathbf{v}_{jl}^c}{(\sigma_j^c)^2} \right\|^2}{\frac{1}{(\sigma_0^c)^2} + \frac{1}{(\sigma_j^c)^2} + \sum_{-j,l} B_{i,l}^{j,c}/(\sigma_j^c)^2} + 2\log \frac{m_i^{-j,c}}{J - m_i^{-j,c}} \\ \quad - \frac{\left\| \boldsymbol{\mu}_0^c/(\sigma_0^c)^2 + \sum_{-j,l} B_{i,l}^{j,c} \mathbf{v}_{jl}^c/(\sigma_j^c)^2 \right\|^2}{1/(\sigma_0^c)^2 + \sum_{-j,l} B_{i,l}^{j,c}/(\sigma_j^c)^2}, \quad i \le L_{-j}^c \ , \\ \frac{\left\| \frac{\boldsymbol{\mu}_0^c}{(\sigma_0^c)^2} + \frac{\mathbf{v}_{jl}^c}{(\sigma_j^c)^2} \right\|^2}{\frac{1}{(\sigma_0^c)^2} + \frac{1}{(\sigma_j^c)^2}} - \frac{\left\| \boldsymbol{\mu}_0^c/(\sigma_0^c)^2 \right\|^2}{1/(\sigma_0^c)^2} - 2\log \frac{i - L_{-j}^c}{\gamma_0/J}, \\ \qquad\qquad L_{-j}^c < i \le L_{-j}^c + L_j^c, \end{cases}$$

where for simplicity we assume $\boldsymbol{\Sigma}_0^c = \boldsymbol{I}(\sigma_0^c)^2$ and $\boldsymbol{\Sigma}_j^c = \boldsymbol{I}(\sigma_j^c)^2$. We then apply the Hungarian algorithm to find the minimizer *of $\sum_i \sum_l B_{i,l}^{j,c} C_{i,l}^{j,c}$ and obtain the neuron matching assignments.*

Proposition 4. *Given the assignment $\{\boldsymbol{B}^{j,c}\}$, the MAP estimate of $\{\theta_i^c\}$ is given by*

$$\hat{\theta}_i^c = \frac{\boldsymbol{\mu}_0^c/(\sigma_0^c)^2 + \sum_{j,l} B_{i,l}^{j,c} \mathbf{v}_{jl}^c/(\sigma_j^c)^2}{1/(\sigma_0^c)^2 + \sum_{j,l} B_{i,l}^{j,c}/(\sigma_j^c)^2} \text{ for } i = 1, \ldots, L. \tag{11}$$

We combine these propositions and summarize the overall multilayer inference procedure in Algorithm 1 in Supplement section 2.

3.3. Neural Matching with Additional Communications

In the traditional federated learning scenario, where local and global models are learned together, common approach (see e.g., McMahan et al. (2017)) is to learn via rounds of communication between local and global models. Typically, local model parameters are trained for few epochs, sent to server for updating the global model and then reinitialized with the global model parameters for the new round. One of the key factors in federated learning is the number of communications required to achieve accurate global model. In the preceding sections we proposed Probabilistic Federated Neural Matching (PFNM) to aggregate local models in a single communication round. Our approach can be naturally extended to benefit from additional communication rounds as follows.

Let t denote a communication round. To initialize local models at round $t+1$ we set $\mathbf{v}_{jl}^{t+1} = \sum_i B_{i,l}^{j,t} \theta_i^t$. Recall that $\sum_i B_{i,l}^{j,t} = 1 \ \forall l = 1, \ldots, L_j, \ j = 1, \ldots, J$, hence a local model is initialized with a *subset* of the global model, keeping local model size L_j constant across communication rounds (this also holds for the multilayer case). After local models are updated we proceed to apply matching to obtain new global model. Note that global model size can change across communication rounds, in particular we expect it to shrink as local models improve on each step.

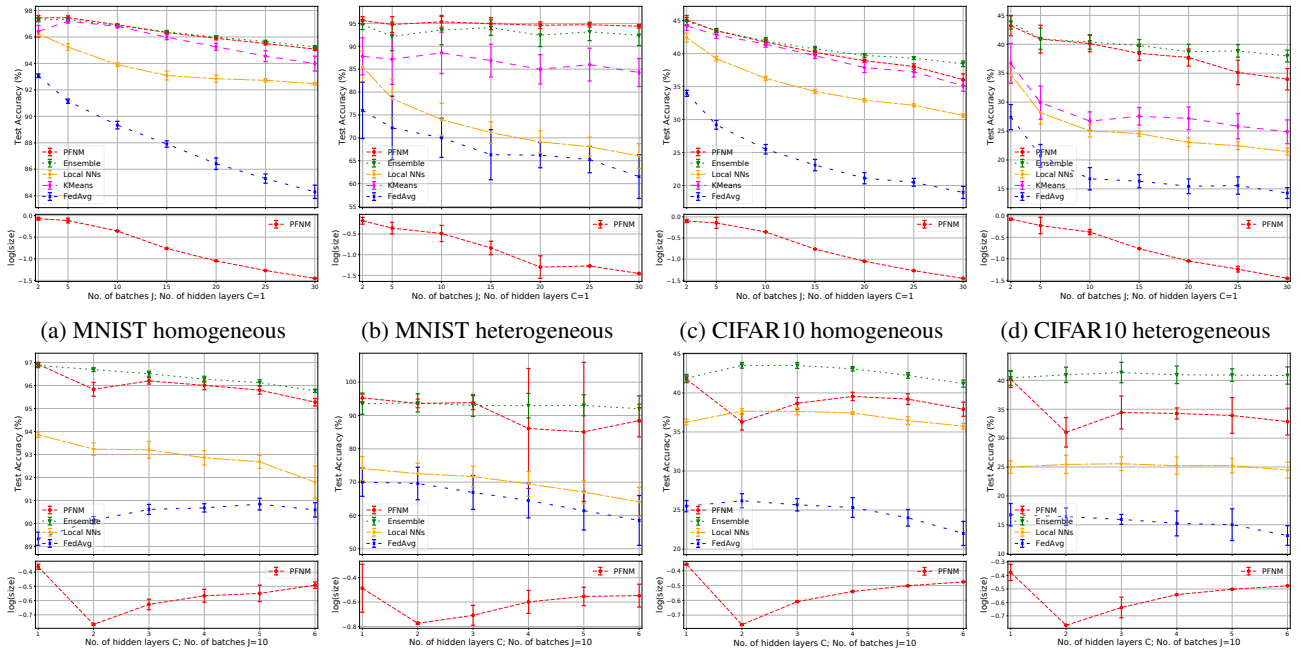
4. Experiments

To verify our methodology we simulate federated learning scenarios using two standard datasets: MNIST and CIFAR-10. We randomly partition each of these datasets into J batches. Two partition strategies are of interest: (a) a homogeneous partition where each batch has approximately equal proportion of each of the K classes; and (b) a heterogeneous partition for which batch sizes and class proportions are unbalanced. We simulate a heterogeneous partition by simulating $\mathbf{p}_k \sim \text{Dir}_J(0.5)$ and allocating a $\mathbf{p}_{k,j}$ proportion of the instances of class k to batch j. Note that due to the small concentration parameter (0.5) of the Dirichlet distribution, some sampled batches may not have any examples of certain classes of data. For each of the four combinations of partition strategy and dataset we run 10 trials to obtain mean performances with standard deviations.

In our empirical studies below, we will show that our framework can aggregate multiple local neural networks trained independently on different batches of data into an efficient, modest-size global neural network with as few as a single communication round. We also demonstrate enhanced performance when additional communication is allowed.

Learning with single communication First we consider a scenario where a global neural network needs to be constructed with a single communication round. This imitates the real-world scenario where data is no longer available and we only have access to pre-trained local models (i.e. "legacy" models). To be useful, this global neural network needs to outperform the individual local models. Ensemble methods (Dietterich, 2000; Breiman, 2001) are a classic approach for combining predictions of multiple learners. They often perform well in practice even when the ensemble members are of poor quality. Unfortunately, in the case of neural networks, ensembles have large storage and inference costs, stemming from having to store and forward propagate through all local networks.

The performance of local NNs and the ensemble method define the lower and upper extremes of aggregating when limited to a single communication. We also compare to other strong baselines, including federated averaging of local neural networks trained with the same random initialization as proposed by McMahan et al. (2017). We note that a federated averaging variant without the shared initialization would likely be more realistic when trying to aggregate pre-trained models, but this variant performs significantly worse than all other baselines. We also consider k-Means



(a) MNIST homogeneous (b) MNIST heterogeneous (c) CIFAR10 homogeneous (d) CIFAR10 heterogeneous

(e) MNIST homogeneous (f) MNIST heterogeneous (g) CIFAR homogeneous (h) CIFAR heterogeneous

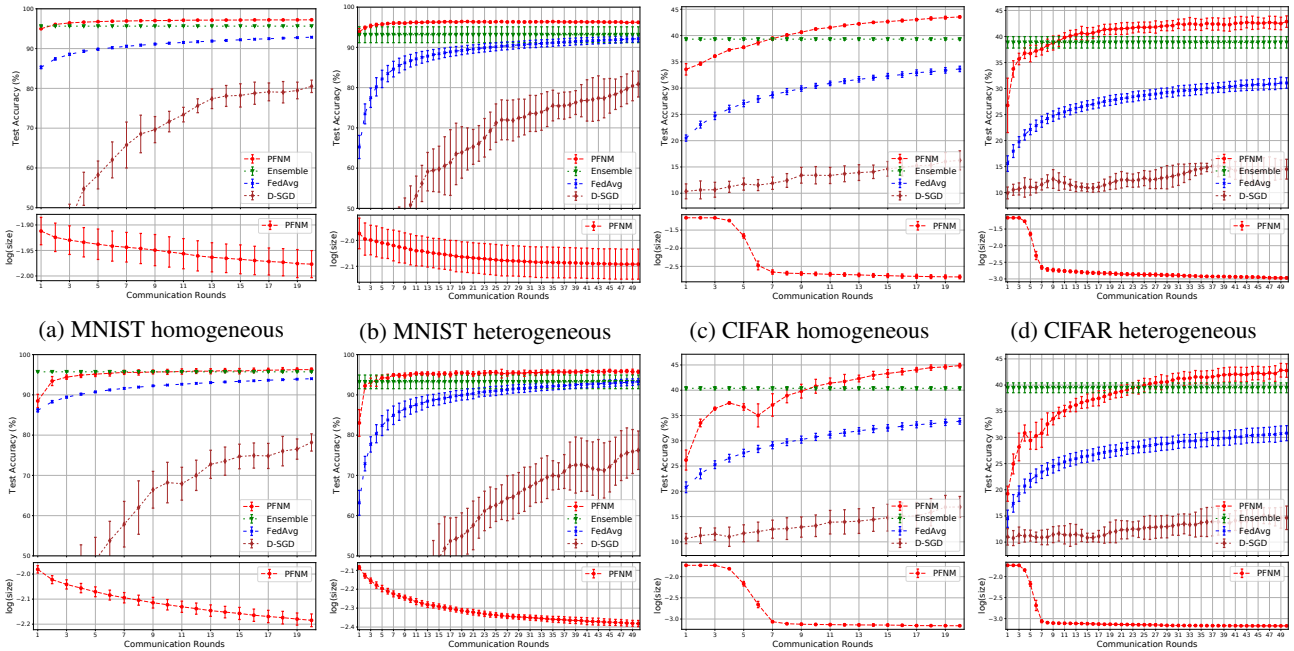
Figure 2: **Single communication federated learning**. TOP: Test accuracy and normalized model size ($\log \frac{L}{\sum_j L_j}$) as a function of varying number of batches (J). BOTTOM: Test accuracy and normalized model size for multi-layer networks as a function of number of layers. PFNM consistently outperforms local models and federated averaging while performing comparably to ensembles at a fraction of the storage and computational costs.

clustering (Lloyd, 1982) of vectors constructed by concatenating weights and biases of local neural networks. The key difference between k-Means and our approach is that clustering, unlike matching, allows several neurons from a single neural network to be assigned to the same global neuron, potentially averaging out their individual feature representations. Further, k-Means requires us to choose k, which we set to $K = \min(500, 50J)$. In contrast, PFNM nonparametrically learns the global model size and other hyperparameters, i.e. $\sigma, \sigma_0, \gamma_0$, are chosen based on the training data. We discuss parameter sensitivity in section three of the Supplement.

Figure 2 presents our results with single hidden layer neural networks for varying number of batches J. Note that a higher number of batches implies fewer data instances per batch, leading to poorer local model performances. The upper plots summarize test data accuracy, while the lower plots show the model size compression achieved by PFNM. Specifically we plot $\log \frac{L}{\sum_j L_j}$, which is the log ratio of the PFNM global model size L to the total number of neurons across all local models (i.e. the size of an ensemble model). In this and subsequent experiments each local neural network has $L_j = 100$ hidden neurons. We see that PFNM produces strong results, occasionally even outperforming ensembles. In the heterogeneous setting we observe a noticeable degradation in the performance of the local NNs and of k-means, while PFNM retains its good performance. It is worth noting that the gap between PFNM and ensemble increases on CIFAR10 with J, while it is constant (and even in favor of PFNM) on MNIST. This is not surprising. Ensemble methods are known to perform particularly well at aggregating "weak" learners (recall higher J implies smaller batches) (Breiman, 2001), while PFNM assumes the neural networks being aggregated already perform reasonably well.

Next, we investigate aggregation of multi-layer neural networks, each using a hundred neurons per layer. The extension of k-means to this setting is unclear and k-means is excluded from further comparisons. In Figure 2, we show that PFNM again provides drastic and consistent improvements over local models and federated averaging. It performs marginally worse than ensembles, especially for deeper networks on CIFAR10. This aligns with our previous observation — when there is insufficient data for training good local models, PFNM's performance marginally degrades with respect to ensembles, but still provides significant compression over ensembles.

Learning with limited communication While in some scenarios limiting communication to a single communication round may be a hard constraint, we also consider sit-



(a) MNIST homogeneous (b) MNIST heterogeneous (c) CIFAR homogeneous (d) CIFAR heterogeneous

(e) MNIST homogeneous (f) MNIST heterogeneous (g) CIFAR homogeneous (h) CIFAR heterogeneous

Figure 3: **Federated learning with communications**. Test accuracy and normalized model size as a function of number of communication rounds for $J = 25$ batches for one (TOP) and two layer (BOTTOM) neural networks. PFNM consistently outperforms strong competitors.

uations, that frequently arise in practice, where a limited amount of communication is permissible. To this end, we investigate federated learning with $J = 25$ batches and up to twenty communications when the data has a homogeneous partition and up to fifty communications under a heterogeneous partition. We compare PFNM, using the communication procedure from Section 3.3 ($\sigma = \sigma_0 = \gamma_0 = 1$ across experiments) to federated averaging and the distributed optimization approach, downpour SGD (D-SGD) of Dean et al. (2012). In this limited communication setting, the ensembles can be outperformed by many distributed learning algorithms provided a large enough communication budget. An interesting metric then is the number of communications rounds required to outperform ensembles.

We report results with both one and two layer neural networks in Figure 3. In either case, we use a hundered neurons per layer. PFNM outperforms ensembles in all scenarios given sufficient communications. Moreover, in all experiments, PFNM requires significantly fewer communication rounds than both federated averaging and D-SGD to achieve a given performance level. In addition to improved performance, additional rounds of communication allow PFNM to shrink the size of the global model as demonstrated in the figure. In Figures 3a to 3f we note steady improvement in accuracy and reduction in the global model size. In CIFAR10 experiments, the two layer PFNM network's performance temporarily drops, which corresponds to a sharp reduction in the size of the global network. See Figures 3g and 3h.

5. Discussion

In this work, we have developed methods for federated learning of neural networks, and empirically demonstrated their favorable properties. Our methods are particularly effective at learning compressed federated networks from pretrained local networks and with a modest communication budget can outperform state-of-the-art algorithms for federated learning of neural networks. In future work, we plan to explore more sophisticated ways of combining local networks especially in the regime where each local network has very few training instances. Our current matching approach is completely unsupervised – incorporating some form of supervision may help further improve the performance of the global network, especially when the local networks are of poor quality. Finally, it is of interest to extend our modeling framework to other architectures such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). The permutation invariance necessitating matching inference also arises in CNNs since any permutation of the filters results in the same output, however additional bookkeeping is needed due to the pooling operations.

References

Breiman, L. Random forests. *Machine learning*, 45(1): 5–32, 2001.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pp. 1223–1231, 2012.

Dietterich, T. G. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pp. 1–15. Springer, 2000.

EU. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119:1–88, may 2016. URL `http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L:2016:119:TOC`.

Ghahramani, Z. and Griffiths, T. L. Infinite latent feature models and the Indian buffet process. In *Advances in Neural Information Processing Systems*, pp. 475–482, 2005.

Griffiths, T. L. and Ghahramani, Z. The Indian buffet process: An introduction and review. *Journal of Machine Learning Research*, 12:1185–1224, 2011.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Kuhn, H. W. The Hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97, 1955.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature*, 521(7553):436, 2015.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 14, pp. 583–598, 2014.

Lian, X., Huang, Y., Li, Y., and Liu, J. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pp. 2737–2745, 2015.

Lian, X., Zhang, C., Zhang, H., Hsieh, C.-J., Zhang, W., and Liu, J. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems 30*, pp. 5330–5340. 2017.

Lloyd, S. Least squares quantization in PCM. *Information Theory, IEEE Transactions on*, 28(2):129–137, Mar 1982.

Ma, C., Smith, V., Jaggi, M., Jordan, M., Richtarik, P., and Takac, M. Adding vs. averaging in distributed primal-dual optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 1973–1982, 2015.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282, 2017.

Moritz, P., Nishihara, R., Stoica, I., and Jordan, M. I. Sparknet: Training deep networks in spark. *arXiv preprint arXiv:1511.06051*, 2015.

Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., and Liao, Q. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, 14(5):503–519, 2017.

Shamir, O., Srebro, N., and Zhang, T. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pp. 1000–1008, 2014.

Smith, V., Chiang, C.-K., Sanjabi, M., and Talwalkar, A. S. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pp. 4424–4434, 2017.

Teh, Y. W., Grür, D., and Ghahramani, Z. Stick-breaking construction for the Indian buffet process. In *Artificial Intelligence and Statistics*, pp. 556–563, 2007.

Thibaux, R. and Jordan, M. I. Hierarchical Beta processes and the Indian buffet process. In *Artificial Intelligence and Statistics*, pp. 564–571, 2007.

Yang, T. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pp. 629–637, 2013.

Yurochkin, M., Fan, Z., Guha, A., Koutris, P., and Nguyen, X. Scalable inference of topic evolution via models for latent geometric structures. *arXiv preprint arXiv:1809.08738*, 2018.

Zhang, Y. and Lin, X. Disco: Distributed optimization for self-concordant empirical loss. In *International conference on machine learning*, pp. 362–370, 2015.

Zhang, Y., Duchi, J., Jordan, M. I., and Wainwright, M. J. Information-theoretic lower bounds for distributed statistical estimation with communication constraints. In *Advances in Neural Information Processing Systems*, pp. 2328–2336, 2013.