Consensus Based Neural Networks With Both Data and Model Partitions

Author Name1 AN1@SAMPLE.COM
Author Name2 AN2@SAMPLE.COM
Author Name3 AN3@SAMPLE.COM

Editors: XXX

Abstract

With rapid advances in technology, storing large volumes of data and using compute power to analyze them has become commonplace. Applications involving sensors, for example, capture unprecedented volumes of data, in different modalities including image, video, audio, GPS and others. Novel algorithms have been designed to deal with this rich data, the most popular being deep neural networks. In this paper, we develop *consensus* based neural network models which learn the network in a distributed setting first using data on the devices to build local models and then communicating with other devices to learn the global model. These algorithms are fundamentally more robust because there is no single point of failure such as a centralized server or a master node in a parallel computing environment. Our algorithm to learn consensus based models uses multilayer perceptrons and rely on both data and model partitions. Each site learns a feedforward neural network and obtains a loss on data stored locally. It then gossips with a neighbor, chosen uniformly at random, and exchanges information on the loss. The new loss is used to run a back propagation algorithm and update weights appropriately. This unique method of feed-forward learning, exchanging losses and back-propagating enables nodes to learn the *global* function without exchange of data in the distributed network. Empirical results on several real world datasets including sensor data reveal that the consensus algorithm has results comparable to centralized multi-layer perceptrons and tree-based algorithms including random forests and gradient boosted decision trees.

Keywords: consensus neural networks, data partition, model partitioning, gossip, push sum

1. Introduction

Emerging architectures designed to store large volumes of data and analyze them make use of large scale, distributed processing paradigms (Provost and Fawcett (2013); Bekkerman et al. (2011)). These include the mega-scale cloud data-centers and the increasingly popular resource constrained devices, such as the Internet of Things (IoT) and mobile devices. While the cloud can be used for executing large scale machine learning algorithms on huge volumes of data, such algorithms exert severe demands in terms of energy, memory and computing resources, limiting their adoption for resource constrained, network edge devices. The new breed of intelligent devices and high-stake applications (drones, augmented/virtual reality, autonomous systems, etc.), require a novel paradigm change calling for distributed, low-

latency and reliable machine learning at the wireless network edge. Thus computing services have now started to move from the cloud to the edge.

Deep learning-based intelligent services (Wang et al. (2020); Zhou et al. (2019); Park et al. (2019)) and applications have become prevalent for both cloud computing and resource constrained environments. They have been used extensively for speech recognition, image and text classification with state-of-the-art performances. However their use in a wider range of applications such as smart cities, drones, and Internet of Vehicles (IoV), has been somewhat limited due to the following reasons: (a) Cost: Training and inference of deep learning models in the distributed infrastructures requires devices or users to transmit massive amounts of data amongst nodes, thus consuming a large amount of network bandwidth. (b) Latency: The access to data and services is generally not guaranteed and delay might not be short enough to satisfy the requirements of many time-critical applications such as cooperative autonomous driving (c) Reliability: Most distributed computing applications rely on wireless communications and backbone networks for connecting users to services, but intelligent services must be highly reliable, even when network connections are lost; (d) Privacy: The data required for deep learning might carry a lot of private information, and privacy issues are critical to areas such as smart home and cities. Thus, the current state of distributed deep learning systems on the cloud and end devices leaves much to be desired.

In this paper, we address this short coming by developing algorithms for learning neural network models on the edge devices by making use of both data and model partitioning. Our algorithms are designed such that the local data are first used for construction of the model using feedforward learning. Then they communicate with one another exchanging information. This exchange of information is facilitated by well studied gossip based protocols (Kempe et al. (2003)) and fundamentally does not require exchange of data amongst the devices. We show that it suffices to exchange loss vectors which then enables the local models to be updated for the back-propagation round. Our algorithms make use of asynchronous Stochastic Gradient Descent (SGD) methods and provide performance comparable to centralized algorithms where-in all of the data is available for model training.

This paper is organized as follows: Section 2 introduces data and model partitions and Section 3 describes applications where consensus based neural networks can be used in practice; Section 4 describes the details of the distributed algorithm for learning multilayer perceptrons followed by discussions in Section 5; Section 6 presents empirical results and Section 7 describes related work in the area. Section 8 concludes the paper and discusses future work required to design consensus based neural network applications.

2. Data and model partitions for neural networks

In large scale distributed environments, the neural network training process contributes significantly to latency and impacts the inference reliability, while delimiting the overall scalability. Thus recent techniques for learning neural networks on edge devices focus on methods of distributing this process thereby enabling the exploitation of more samples collected from the environment. Conceptually, the training process can be split up (Park et al. (2019)) according to the following:

1. Data Split: Parallelizing the data samples to multiple devices that have an identical structure

2. Model Split: When a neural network model size is too large, a single neural network structure can be split into multiple segments that are distributed over multiple devices

In this paper, we explore a different architecture for training using consensus – the data split occurs by partitioning the feature space so that devices encounter only a subset of features for all the instances (for instance, the features could be locked in time) and each device learns identical structures. Hypothetically, the process can be viewed as splitting of a very large centralized neural network model into multiple segments and learnt individually on devices; while most model split training processes depend on a master slave architecture for coordination amongst the different parts, our algorithm uses a gossip based algorithm which is completely decentralized and depends on peer-to-peer communication.

3. Use cases for learning consensus based neural networks

We motivate the need to develop consensus based neural networks with both data and model partitions by describing the following applications: (a) **Medical Diagnosis:** Collaborations amongst health entities (Gupta and Raskar (2018b); Vepakomma et al. (2018)) require examination of different modalities of patient data such as electronic health records (EHR), picture archiving and communication systems (PACS) for radiology and other imaging data, pathology test results, and other sensitive data such as genetic markers for disease. These scenarios require training of distributed machine learning models (such as neural networks) with little or no data sharing. (b) **Autonomous Vehicles:** Google, Uber, Tesla, Mobileye and many automotive companies have developed autonomous driving systems (Lin et al. (2018); Abdel-Aziz et al. (2018)). Such a system allows the vehicle to drive by itself without requiring help from a human. The vehicle is equipped with autonomous driving capability detects the environment, locates its position, and operates the vehicle to get to the destination safely without human input. Vehicle-to-vehicle (V2V) safety applications are crucial for autonomous driving systems and include forward collision warning, blind spot, lane change warnings, and adaptive cruise control. These applications are time critical and require real time updates from individual vehicles. (c) **Home Sensing:** In home monitoring and sensing applications, environmental sensors and IoT devices detect fluctuations in signals. Heurta et al. (Huerta et al. (2016b)) study methods for online de-correlation of chemical sensor signals from the effects of environmental humidity and temperature variations; non-intrusive load monitoring systems (Batra et al. (2014)) are used to study fluctuations in household energy consumption data collected from several applications. The above use cases provide examples of the use of sophisticated machine learning techniques such as consensus based neural networks.

4. Distributed Neural Network Algorithms

We present the consensus based neural network algorithm in this section.

In the distributed setting, let M denote an $N \times n$ matrix with real-valued entries. This matrix represents a dataset of N tuples of the form $x_i \in \mathbb{R}^n, 1 \leq i \leq N$. Each tuple has an associated label $y_i = \{+1, -1\}$. Assume this dataset has been *vertically*[1] distributed

1. This implies that all the sites have access to all N tuples but have limited number of features i.e. $n_i \subset n$.

over m sites $S_1, S_2, \cdots, S_m$ such that site S_i has a data set $M_i \subset M, M_i : N \times n_i$ and each $x_j \in M_i$ is in $\mathbb{R}^{n_i}, n_i \subset n$. Thus, $M = M_1 \cup M_2 \cup \cdots \cup M_m$ denotes the concatenation of the local datasets. The labels are shared across all the sites. The goal is to learn a deep neural network on the global data set M, by learning local models[2] at the sites, allowing exchange of information among them using a gossip based protocol (Kempe et al. (2003); Demers et al. (1987)) and updating the local models with new information obtained from neighbors. This ensures that there is no actual data transfer amongst sites.

Model of Distributed Computation. The distributed algorithm evolves over discrete time with respect to a "global" clock[3]. Each site has access to a local clock or no clock at all. Furthermore, each site has its own memory and can perform local computation (such as estimating the local weight vector). It stores f_i, which is the estimated local function. Besides its own computation, sites may receive messages from their neighbors which will help in evaluation of the next estimate for the local function.

Communication Protocols. Sites S_i are connected to one another via an underlying communication framework represented by a graph $G(V, E)$, such that each site $S_i \in \{S_1, S_2, \cdots, S_m\}$ is a vertex and an edge $e_{ij} \in E$ connects sites S_i and S_j. Communication delays on the edges in the graph are assumed to be zero. It must be noted that the communication framework is usually expected to be application dependent. In cases where no intuitive framework exists, it may be possible to simply rely on the physical connectivity of the machines, for example, if the sites S_i are part of a large cluster.

We present next, two algorithms for binary classification:

1. **Distributed Sigmoid Threshold Unit (DSTU)**, which is the simplest neural network that can be learnt at a site and
2. **Distributed MultiLayer Perceptron (DMLP)** algorithm.

Distributed Sigmoid Threshold Unit (DSTU): Assume that the feature vector at a site S_i is processed by the sigmoid threshold unit i.e. $\hat{y} = \sigma(w^T x), \sigma(y) = \frac{1}{1+e^{-y}}$ and η is the learning parameter. The unit updates the parameter w at iteration t at a local site using the following rule: $w_{t+1} = w_t + \eta y_t x_t$ where $w_t, w_{t+1}, x_t \in R^{n_i}$ and we consider the hinge loss function. Figure 1 describes the algorithm for a global consensus on weights at all sites in the network. Each site learns a model on its own local data and exchanges loss vectors with neighbors. The new loss is used to update local weight vectors.

Distributed MultiLayer Perceptron (DMLP): Assume that each site S_t has a simple model of a fully connected feed-forward deep neural network with a single output and rectified linear units. The network is called $\mathcal{N}_t$. It has L layers – the 0^{th} is the input layer, followed by $(L-1)$ hidden layers and the L^{th} layer is the output layer. Let r_i denote the number of units in the i^{th} layer (note that $r_0 = n_i$ and $r_L = 1$).

Feedforward Learning: Let ω_{ij}^k denote the weight from i^{th} node of $(k-1)^{th}$ layer to j^{th} node of k^{th} layer, a_j^k is the weighted sum of inputs from the previous layer to the j^{th} node of k^{th} layer, o_j^k is the output of j^{th} node of k^{th} layer, b_j^k is the bias to j^{th} node of k^{th} layer. The feed-forward step (Figure 1) for the first node of the first hidden layer can then

2. We assume that the models have the same structure i.e. the same number of input, hidden and output layers and connections.
3. Existence of this clock is of interest only for theoretical analysis

Input: $N \times n_i$ matrix at each site S_i, $G(V, E)$ which encapsulates the underlying communication framework, T : no of iterations
Output: Each site S_i has $W_i \approx W_g[1 : n_i]$

for *t = 1 to T* **do**
(a) Site S_i computes W_i locally and estimates the loss on N instances
(b) Site S_i gossips with its neighbors S_j and obtains the loss from the neighbor
(c) **Gossip:** Site S_i averages the loss between S_i and S_j and sets this as the new loss
(d) Update the local weight vectors using stochastic gradient descent and the new loss obtained after gossiping
(e) If there is no significant change in the local weight vectors of one of the sites then stop
end

Algorithm 1: Distributed Sigmoid Threshold Unit (DSTU) Learning by Consensus-Based Modeling

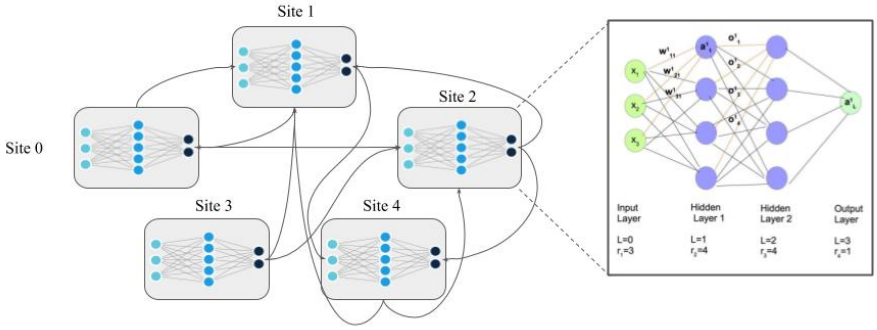

Figure 1: (Left) The distributed neural network. (Right) Illustration of the notation used to represent a Multilayer Perceptron with an input, two hidden and one output layer.

be written as: $a_1^1 = b_1^1 + x_1^0 * \omega_{11}^1 + x_2^0 * \omega_{21}^1 + ... + x_{n_i 1}^0 * \omega_{n_i 1}^1$. Assuming sigmoid activation, the output of node a_1^1 is given by: $o_1^1 = \sigma(a_1^1)$. So, the output of j^{th} node of k^{th} layer is, $o_j^k = \sigma(a_j^k)$ where, $a_j^k = b_j^k + (\sum_{i=1}^{r_{k-1}} o_i^{k-1} * w_{ij}^k)$. The output from the network $\mathcal{N}_t$ is given by $\hat{y}_i^{\mathcal{N}_t} = \sigma(a_1^L)$. The local loss at site S_i is then given by $\mathcal{L}_t = \frac{1}{2} \sum_{i=1}^{N} (y_i - \hat{y}_i^{\mathcal{N}_t})^2$, assuming squared loss.

Gossip: Site S_t selects uniformly at random, a neighbor S_u with whom it wishes to gossip. Both S_t and S_u have computed their local losses. When gossiping each site updates its current local loss with $\mathcal{L}_{gossip} = \frac{\mathcal{L}_t + \mathcal{L}_u}{2}$. This new loss is used for backpropagation at both sites S_t and S_u.

Backpropagation: The Backpropagation algorithm learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. It employs gradient descent to attempt to minimize the squared error between the network output values and the

target values for these outputs. We use the new loss ($\mathcal{L}_{gossip}$) obtained after gossiping with a neighbor, in-place of the local loss ($\mathcal{L}_t$), for our backpropagation phase. This modification helps the local site S_t to incorporate information about the loss from its neighbor S_u into its backpropagation learning phase, thereby helping to minimize the *global* loss instead of the local loss. This is a crucial step in our algorithm. The local loss at site S_t after gossip is then given by $\mathcal{L}_{gossip} = \frac{1}{2}\sum_{i=1}^{N}(y_i - \frac{\hat{y}_i^{\mathcal{N}_t} + \hat{y}_i^{\mathcal{N}_u}}{2})^2 = \frac{1}{2}(\mathbf{y} - \mathbf{y_{gossip}})^2; \mathbf{y_{gossip}} = \frac{\hat{y}_i^{\mathcal{N}_t} + \hat{y}_i^{\mathcal{N}_u}}{2}$ where the bold fonts are used to represent the loss vectors. We present next, expressions for $\frac{\partial \mathcal{L}_{gossip}}{\partial \omega_{ij}^k}$, to implement the Stochastic Gradient Descent rule. We consider two cases: (a) when the unit j is an output unit and (b) when the unit j is a hidden unit.

Training rule for output unit weights: The change in loss w.r.t. weight is given by: $\frac{\partial \mathcal{L}_{gossip}}{\partial \omega_{ij}^k} = \frac{\partial \mathcal{L}_{gossip}}{\partial o_j^k} \cdot \frac{\partial o_j^k}{\partial a_j^k} \cdot \frac{\partial a_j^k}{\partial \omega_{ij}^k}$. If, $\frac{\partial \mathcal{L}_{gossip}}{\partial o_j^k} \cdot \frac{\partial o_j^k}{\partial a_j^k} = \sigma_j^k$, then for the final layer L,

$$\frac{\partial \mathcal{L}_{gossip}}{\partial \omega_{ij}^L} = \frac{\partial \mathcal{L}_{gossip}}{\partial o_j^L} \cdot \frac{\partial o_j^L}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial \omega_{ij}^L} \tag{1}$$

Now, $\frac{\partial \mathcal{L}_{gossip}}{\partial o_j^L} = \frac{\partial (\frac{1}{2}(\mathbf{y} - \mathbf{y_{\hat{gossip}}})^2)}{\partial o_j^L} = \frac{\partial (\frac{1}{2}(\mathbf{y} - o_j^L)^2)}{\partial o_j^L} = \frac{1}{2} * 2 * (\mathbf{y} - o_j^L) = (\mathbf{y} - \hat{\mathbf{y}}_{\mathbf{gossip}})$. Again, $\frac{\partial o_j^L}{\partial a_j^L} = \frac{\partial \sigma(a_j^L)}{\partial a_j^L} = \sigma'(a_j^L)$. For sigmoid function, $\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$ so, $\frac{\partial o_j^L}{\partial a_j^L} = \sigma(a_j^L) \cdot (1 - \sigma(a_j^L))$ whereby, the error term for final layer is calculated as,

$$\delta_j^L = (\mathbf{y} - \mathbf{y_{\hat{gossip}}}) \cdot \sigma(a_j^L) \cdot (1 - \sigma(a_j^L) \tag{2}$$

Since the final layer has only one node, j = 1. The weights coming to the output layer $\partial(b_1^L + o_1^{L-1} \cdot \omega_{11}^L + o_2^{L-1} \cdot \omega_{21}^L + ..+$ can be represented as: $\frac{\partial a_j^L}{\partial \omega_{ij}^L} = \frac{o_{r_{L-1}}^{L-1} \cdot \omega_{r_{L-1}1}^L)}{\partial \omega_{ij}^L} = o_i^{L-1}$. Substituting in Equation 1, $\frac{\partial \mathcal{L}_{gossip}}{\partial \omega_{ij}^L} = (\mathbf{y} - \mathbf{y_{\hat{gossip}}}) \cdot \sigma(a_j^L) \cdot (1 - \sigma(a_j^L)) \cdot o_i^{L-1}$. This can be written as: $\frac{\partial \mathcal{L}_{gossip}}{\partial \omega_{ij}^L} = (y - o_j^L) \cdot (o_j^L) \cdot (1 - o_j^L)) \cdot o_i^{L-1}$.

Training rule for hidden unit weights: For hidden layer k, we have:

$$\frac{\partial \mathcal{L}_{gossip}}{\partial \omega_{ij}^k} = \frac{\partial \mathcal{L}_{gossip}}{\partial o_j^k} \cdot \frac{\partial o_j^k}{\partial a_j^k} \cdot \frac{\partial a_j^k}{\partial \omega_{ij}^k} \tag{3}$$

Also, $\frac{\partial \mathcal{L}_{gossip}}{\partial o_j^k} = \sum_{l=1}^{n_{k+1}} \frac{\partial \mathcal{L}_{gossip}}{\partial o_l^{k+1}} \cdot \frac{\partial o_l^{k+1}}{\partial a_l^{k+1}} \cdot \frac{\partial a_l^{k+1}}{\partial o_j^k}$ and $\frac{\partial \mathcal{L}_{gossip}}{\partial o_l^{k+1}} \cdot \frac{\partial o_l^{k+1}}{\partial a_l^{k+1}} = \delta_l^{k+1}$. Substituting in equation 3, we have:

$\frac{\partial \mathcal{L}_{gossip}}{\partial o_j^k} = \sum_{l=1}^{n_{k+1}} \delta_l^{k+1} \cdot \frac{\partial a_l^{k+1}}{\partial o_j^k}$

Now, $\frac{\partial a_l^{k+1}}{\partial o_j^k} = \omega_{jl}^{k+1}$ which implies, $\frac{\partial \mathcal{L}_{gossip}}{\partial o_j^k} = \sum_{l=1}^{n_{k+1}} \delta_l^{k+1} \cdot \omega_{jl}^{k+1}$. Also, $\frac{\partial a_j^k}{\partial \omega_{ij}^k} = o_i^{k-1}$ and $\frac{\partial o_j^k}{\partial a_j^k} = \sigma'(a_j^k) = \sigma(a_j^k) \cdot (1 - \sigma(a_j^k)) = (o_i^k) \cdot (1 - o_i^k)$. Putting it all together,

$$\frac{\partial \mathcal{L}_{gossip}}{\partial \omega_{ij}^k} = \sum_{l=1}^{n_{k+1}} \delta_l^{k+1} \cdot \omega_{jl}^{k+1} \cdot o_i^{k-1} \cdot (o_i^k) \cdot (1 - o_i^k) \tag{4}$$

Now the bias term can be written as $b_j^k = \omega_{0j}^k$ whereby Equation 4 provides the generalized formula for error change w.r.t weight ω_{ij}^k. Simplifying further, δ_j^k is given by, $\delta_j^k = \frac{\partial \mathcal{L}_{gossip}}{\partial o_j^k} \cdot \frac{\partial a_j^k}{\partial a_j^k} = \sum_{l=1}^{n_{k+1}} \delta_l^{k+1} \cdot \omega_{jl}^{k+1} \cdot o_i^k \cdot (1 - o_i^k)$ and $\frac{\partial \mathcal{L}_{gossip}}{\partial \omega_{ij}^k} = \delta_j^k \cdot (o_i^{k-1})$. Algorithm 2 presents the steps of the DMLP algorithm.

Input: $N \times n_i$ matrix at each site S_i, $G(V, E)$ which encapsulates the underlying communication framework, T : no of iterations
Output: Each site S_i has a multilayer perceptron network $\mathcal{N}_i$

for *t = 1 to T* **do**
(a) Site S_i uses the network $\mathcal{N}_i$ for feedforward learning and locally estimates the loss on N instances
(b) Site S_i gossips with its neighbors S_j and obtains the loss from the neighbor
(c) **Gossip:** Site S_i averages the loss between S_i and S_j and sets this as the new loss
(d) Perform backpropagation on the current site and the neighbor site using the gossiped loss; Update the weight vectors in each layer using Stochastic Gradient Descent (SGD)
(e) If there is no significant change in the local weight vectors, stop.
end

Algorithm 2: Distributed Multilayer Perceptron Learning (DMLP)

5. Discussion

Some interesting aspects of our distributed deep neural network algorithms are as follows: (a) **Anytime Algorithms:** The algorithms presented in the above section are called anytime algorithms (Zilberstein (1993); Zilberstein and Russell (1993)). Anytime algorithms are those whose quality of results change gradually as computation time increases. They extend the traditional notion of a computational procedure by allowing it to return many possible approximate answers to any given input. The notion of approximate processing and the use of principles of bounded rationality have proved useful in these applications. What is special about anytime algorithms is the use of well-defined quality measures to monitor the progress in problem solving and allocate computational resources effectively. (b) **Effect of different loss functions and activations:** The derivation presented above uses squared loss and a sigmoid activation function. This can be extended for other kinds of loss functions such as cross-entropy, softmax etc. and activations such as linear, tanh, etc. For example, with the cross-entropy loss function we have: $\frac{\partial \mathcal{L}_{gossip}}{\partial o_j^L} = \frac{\partial((1-y)\log(1-\hat{y_{gossip}}))}{\partial o_j^L} = -\frac{(y/o_j^L - (1-y)}{(1-o_j^L))} = \frac{(o_j^L - y)}{((1-o_j^L)o_j^L)}$ (c) **Effect of variation in number of *deep* layers and output:** The number of hidden layers of the neural network algorithm can be incremented as required by a site, without the need for any algorithmic changes. Similarly, while theoretical results have been demonstrated above with one output layer, there are no changes in the behavior of the algorithm when the number of output layers increases.

6. Empirical Results

The purpose of our empirical results is to demonstrate the utility of the distributed multilayer perceptron algorithm (DMLP) (described in Section 4). We intend to examine the following questions:

1. Is there empirical support for the conjecture that the performance of the *Distributed* model is better than that of the *Centralized* model?

2. How does the performance of the proposed consensus based neural network model compare to state-of-the-art supervised learning methods such as Random Forests (Breiman (2001)) and tree boosting algorithms (such as XGBoost (Chen and Guestrin (2016)))?

The answers to the above questions were explored using datasets curated as part of the NIPS 2003 feature selection challenge (Guyon et al. (2004)) along with a real world IoT dataset (Huerta et al. (2016a)). A brief description follows:

1. **Arcene:** Distinguishes cancer versus normal patterns from mass-spectrometric data.

2. **Dexter:** A text classification problem in a sparse bag-of-word representation, created from a subset of the Reuters dataset.

3. **Dorothea Bal.** : Drug discovery data in which chemical compounds represented by structural molecular features must be classified as active (binding to thrombin) or inactive.

4. **Gisette :** Handwriting recognition data created from the MNIST dataset (LeCun and Cortes (2010)), by selecting two digits "4" and "9".

5. **Madelon:** Artificial data where the problem is multivariate and highly nonlinear.

6. **MNIST Bal.:** Data pertaining to handwritten digits from 0 to 9. It was converted into a binary classification problem by selecting 8 to be the positive label, and the remaining digits to be negative.The negative class is downsampled to contain the same number of samples as the positive class.

7. **HT Sensor:** The dataset contains recordings from a gas sensor array composed of 8 MOX gas sensors, and a temperature and humidity sensor. This sensor array was exposed to background home activity while subject to two different stimuli: wine and banana. This is an IOT dataset with a considerable number of data points.

A summary of the size of the train and test datasets is shown in Table 1. The experimental process is described below:(a) The Peersim simulator (Montresor and Jelasity (2009)) is used to construct a fully connected graph of 10 sites. Each site can independently store data. (b) The total number of features in the train data is split into 10 roughly equal parts. Each site is assigned the data with the corresponding split containing all the examples but only those features it has been assigned. (c) Each site builds a local neural network model. The local loss vector is generated. (d) Each site selects a neighbor uniformly at random according to the underlying distributed graph, and exchanges the local loss vector with its

Dataset	No. Train	No. Test	No. Features
Arcene	100	100	10000
Dexter	300	300	20000
Dorothea Bal.	156	68	100000
Gisette	6000	1000	5000
Madelon	2000	600	500
MNIST Bal.	11702	1948	784
HT Sensor	14560	3640	10

Table 1: Characteristics of the datasets used for empirical analysis.

neighbor. The new loss vector is computed as the average of its own loss vector and that of the neighbor's. (e) Each site updates the model using back propagation using the new loss generated after gossiping with neighbor. (f) The above process is repeated for several iterations until the sites converge to a solution. (g) Each site is provided with the test set having only those features that the site used to construct the local model. For each test sample, an average predicted value is obtained across all sites, and the distributed test accuracy is obtained after applying a threshold of 0.5 on the average predictions obtained.

We measure the performance of the model by the area under the Receiver Operating Characteristic (ROC) curve (Bradley (1997)) denoted by θ. The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR), for all classification thresholds between 0 and 1. The TPR and FPR are defined as follows:

$$TPR = \frac{TP}{TP + FN}, FPR = \frac{FP}{FP + TN} \tag{5}$$

where TP, TN, FP and FN are respectively the number of true positive, true negatives, false positive and false negatives. The area under the ROC curve (θ) provides an aggregate measure of performance across all possible classification thresholds and is a useful metric to measure the performance of binary classification problems when the best threshold for output predictions is unknown.

The centralized algorithm is executed by assuming that the entire dataset is available at a site. A single-layer neural network is employed at each site with the cross-entropy loss function, sigmoid activation for the hidden layer, and softmax activation for the output layer. The number of hidden neurons is kept the same for both the centralized and distributed experiments. This implies that each distributed site has roughly $\frac{\text{Number of hidden neurons in centralized model}}{\text{Number of sites}}$ hidden neurons in its model. For each dataset, the learning rate and number of hidden neurons in the hidden layer are tuned using a grid search across many parameter configurations.

The steps outlined for the distributed algorithm above was repeated for three random feature splits and the test AUC averaged over the trials. The results are presented in 2. Figure 3 shows the AUC curves for all the datasets used in this study. It can be seen that the DMLP algorithm outperforms the centralized MLP algorithm in the case of Dexter, and achieves comparable performance for all the remaining datasets. We also compute the

Dataset	No. Hidden Neurons (C)	No. Hidden Neurons (D)	Learning Rate	Centralized AUC	Distributed AUC	95% C. I.	Cent. Itr. (I_C)	Dist. Itr. (I_D)
Arcene	50	5	0.01	0.86 ± 0.44	0.83 ± 0.58	$[0.77, 0.88]$	139	119
Dexter	20	2	0.1	0.72 ± 0.48	0.83 ± 0.31	$[0.80, 0.86]$	832.33	783.33
Dorothea Bal.	100	10	0.01	0.89 ± 0.27	0.89 ± 0.3	$[0.83, 0.94]$	139	199
Gisette	200	20	0.001	0.90 ± 0.31	0.88 ± 0.44	$[0.87, 0.90]$	742.33	989
Madelon	50	5	0.05	0.61 ± 0.11	0.60 ± 0.22	$[0.57, 0.62]$	312.33	142.33
MNIST Bal.	100	10	0.01	0.90 ± 0.34	0.89 ± 0.34	$[0.88, 0.90]$	272	312
HT Sensor	20	2	0.1	0.99 ± 0.057	0.89 ± 0.33	$[0.88, 0.90]$	835.67	335.67

Table 2: Performance of the centralized(C) and distributed algorithms(D). The consensus neural network uses cross-entropy loss function, sigmoid activation for the hidden layer, and soft-max activation for the output layer. The results are averaged over three trials. Cent. Itr. refers to the number of iterations to convergence and Dist. Itr. the corresponding number of distributed iterations.

symmetric 95% confidence interval for distributed test AUC (θ_D) and observe centralized test AUC (θ_C) in relation to this interval.

The Standard Error (SE) for estimated area under the ROC curve in relation to the sample size (n) and θ_D can be computed as described in Hanley and Mcneil (1983):

$$SE = \sqrt{\frac{\theta(1-\theta_D) + (n-1)(Q_1 + Q_2 - 2\theta_D^2)}{n^2}}$$

where $Q_1 = \frac{\theta_D}{2-\theta_D}, Q_2 = \frac{2\theta_D^2}{1+\theta_D}$. Given SE, the symmetric 95% confidence interval (CI) is given by $\theta_D \pm 1.96(SE)$. The centralized algorithm and distributed algorithm can be deemed approximately comparable if θ_C lies within these bounds, i.e. if $\theta_D - 1.96(SE) <= \theta_C <= \theta_D + 1.96(SE)$. In empirical studies, it was found that the distributed algorithm obtains comparable test AUC scores to the centralized algorithm for Arcene, Madelon, Gisette and MNIST datasets, after taking into account the variance of the confidence interval generated.

Furthermore, we study the impact of the overlap of features at each site on the performance of the consensus algorithm using the *overlap_ratio* parameter. An *overlap_ratio* of 0 indicates that the features present at one site are not present at any other site, i.e. the feature space is partitioned with mutual exclusivity. On the other hand, an *overlap_ratio* greater than 0, indicates that a subset of the feature space is shared among all sites. Table 5 presents the results of the experiments with *overlap_ratio* set to 0.2.

Our results reveal that for some data sets such as the HT Sensor and Dexter, the overlap of features amongst sites is beneficial in boosting the performance of the consensus algorithm. However, this behavior is not consistent and some datasets such as Gisette do not necessarily show improvement in performance even as overlap amongst features increase.

Finally, given that data partition at each site involves exploring a subset of the feature space, we compared the consensus algorithm to state-of-the-art tree-based algorithms such as Random Forests and XGBoost. The results are presented in Table 4.

We observe that the consensus algorithm does not perform as well as RF and XGBoost in Dexter, Gisette, Madelon, MNIST and HT datasets. However, it must be noted that in the

Dataset	Cent AUC	Dist. w/overlap AUC	95% C. I.	Cent. Itr. (I_C)	Dist. Itr. (I_D)
Arcene	0.86 ± 0.44	0.85 ± 0.4	$[0.79, 0.90]$	139	136
Dexter	0.72 ± 0.48	0.82 ± 0.30	$[0.79, 0.85]$	782.33	474.15
Dorothea Bal.	0.89 ± 0.27	0.88 ± 0.2	$[0.82, 0.94]$	139	155.67
Gisette	0.90 ± 0.31	0.88 ± 0.48	$[0.86, 0.89]$	742.33	929
Madelon	0.61 ± 0.11	0.60 ± 0.78	$[0.57, 0.64]$	312.33	139
MNIST Bal.	0.90 ± 0.34	0.88 ± 0.38	$[0.87, 0.89]$	272	305.67
HT Sensor	0.99 ± 0.057	0.98 ± 0.16	$[0.97, 0.98]$	835.67	335.67

Table 3: Performance of the centralized(C) and distributed algorithms(D) with 0.2% overlap of features. The consensus neural network uses cross-entropy loss function, sigmoid activation for the hidden layer, and soft-max activation for the output layer. The results are averaged over three trials.

Dataset	RF_AUC	XGBoost_AUC	Dist_AUC	Cent_AUC
Arcene	0.79	0.84	0.83	0.86
Dexter	0.93	0.95	0.83	0.72
Dorothea Bal.	0.88	0.89	0.89	0.89
Gisette	0.99	0.99	0.88	0.90
Madelon	0.77	0.71	0.60	0.61
MNIST Bal.	0.99	0.99	0.89	0.90
HT	1	0.99	0.89	0.99

Table 4: Comparison of the performance of the consensus algorithm to tree based algorithms Random Forest (RF) and XGBoost.

consensus and centralized settings we have only studied a multi-layer perception with cross-entropy loss and sigmoid activation function.The architecture was not tuned to obtain the best neural network possible. This set of controlled experiments enabled experimentation with the multilayer perceptron architecture, and the consequences of using it in a large scale consensus setting. The Appendix presents results on using RELU activations, which enhance performance significantly for several datasets. Future work will involve tuning the architecture to obtain the best possible results on a given dataset and explore the effects of a consensus algorithm using that architecture.

7. Related Work

Scalable algorithms for deep learning have been explored in several papers in recent years. We discuss related work which make use of two different architectures: (a) Parallel – which ensures the presence of a master to control slave workers and (b) Distributed which is a fully decentralized, peer-to-peer architecture without the need for a single master.

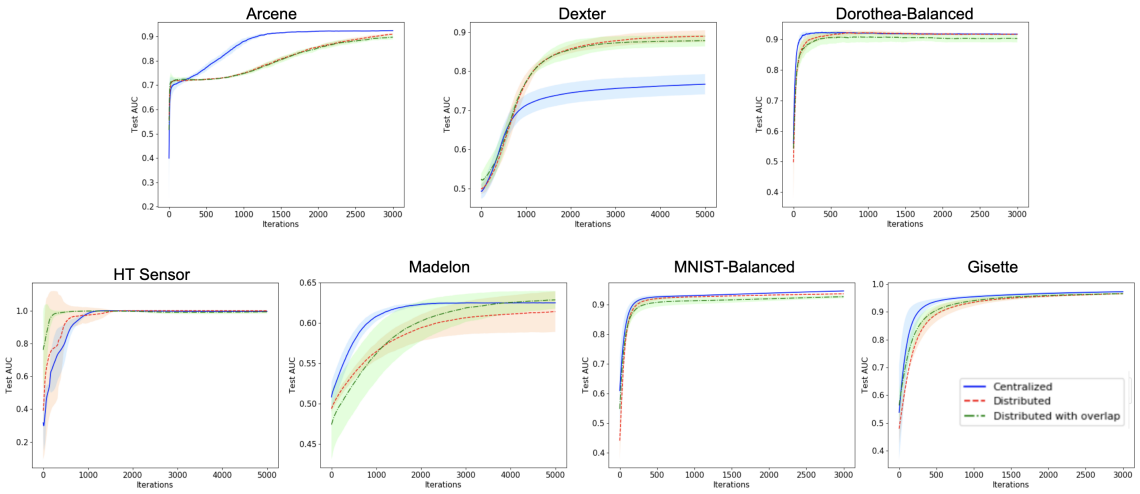

Figure 2: AUC on the test sets for both centralized and distributed settings on the seven datasets discussed above. For the distributed algorithm, test accuracy results averaged over three random vertical feature splits with and without overlap are presented.

Parallel DNN Algorithms: A large proportion of the research in this domain has focused on data parallelism and the ability to exploit compute power of multiple slave workers, with a single master controlling the execution of slaves. McDonald et al. (McDonald et al. (2010)) present two different strategies for parallel training of structured perceptrons and use them for named entity recognition and dependency parsing. TernGrad (Wen et al. (2017)) uses ternary levels $\{-1, 0, +1\}$ to reduce overhead of gradient synchronization and communication. They also include in their algorithm a technique for layer-wise ternarizing and gradient clipping to ensure the performance of the algorithm closely mimics Stochastic Gradient Descent (SGD). DoReFa-Net (Zhou et al. (2016)) train convolutional neural networks that have low bit width weights, activations and gradients. Parameter gradients are stochastically quantized to low bit width numbers before being propagated to convolutional layers. Seide et al. (Seide et al. (2014)) show that it is possible to quantize gradients aggressively during training of deep neural networks using SGD making it feasible to use in data parallel fast processors such as GPUs. Quantized SGD (QSGD) (Alistarh et al. (2017)) explores the trade-off between accuracy and gradient precision. The effectiveness of gradient quantization has been justified and the convergence of QSGD was provably guaranteed. A slightly different line of work (Zhang et al. (2016)) explores the utility of asynchronous Stochastic Gradient Descent algorithms suggesting that if the learning rate is modulated according to the gradient staleness, better theoretical guarantees for convergence can be established than the synchronous counterpart. Blot et al.(Blot et al. (2019)) propose an asynchronous distributed algorithm, GoSGD, to improve the training speed of convolutional neural networks which uses gossip to share weights in a multi-threaded envi-

ronment. Although gossip protocols play an important role in their algorithm design, the architecture does assume the presence of a centralized server much like Federated Learning (McMahan et al. (2017)). The existence of a centralized server is often undesirable in large scale distributed applications since the server tends to become a Single Point Of Failure (SPOF).

Distributed DNN Algorithms: In the fully decentralized setting, Jiang et al. (Jiang et al. (2017)) present a consensus-based distributed SGD (CDSGD) (and its momentum variant, CDMSGD) algorithm for collaborative deep learning over fixed topology networks that enables data parallelization as well as decentralized computation. While this work is closely related to our proposed research, these algorithms have not been studied in the context of simultaneous data and model partitions, which is the main focus of this work.

Model Partition: Sutton et al. (Sutton et al. (2009)) explore neural network architectures in which the structures of the models are partitioned prior to training. They experiment with models which have successively partitioned hidden layers starting from the last hidden layer ensuring that neurons in the left and right halves of a hidden layer have connections only to the left and right halves of the layer above, respectively. Partitioning of deep neural networks have also been studied in the context of distributed computing hierarchies such as the cloud, end and edge devices (Teerapittayanon et al. (2017); Lin et al. (2019); Kang et al. (2017)). In Neurosurgeon (Kang et al. (2017)) the authors examine the state-of-the-art approach of cloud-only processing and investigate computation partitioning strategies that effectively leverage compute cycles in the cloud and on the mobile device to achieve low latency, low energy consumption, and high datacenter throughput for a class of intelligent applications. Gupta et al. (Gupta and Raskar (2018a)) present an algorithm for training DNNs over multiple data sources. They observe that feed forward learning (F) involves sequential application of individual layers of the network on the data – this enables development of an algorithm wherein the first few layers are applied on the data first at site A and then the remaining layers are applied at site B i.e. $F(data)$ is same as $F_B(F_A(data))$. In the software framework called DistBelief (Dean et al. (2012)), two algorithms for distributed learning of deep neural networks are presented – (a) Downpour SGD which uses asynchronous SGD to learn models and (b) Sandblaster L-BFGS which supports several distributed batch optimization techniques. While Downpour SGD does use asynchronous SGD, similar to the setup in this study, the major difference lies in the fact that Downpour SGD uses a single parameter server to update billions of parameters of the models in a sharded environment.

Finally, the studies described above fundamentally differ from the material presented in this paper in that our consensus algorithm relies on both model and data partitioning to construct local neural network models which can independently learn global information.

8. Conclusion and Future Work

This paper presents an algorithm for learning consensus based multilayer perceptrons in cloud computing environments as well as on edge devices such as sensors and mobile devices. The algorithm is developed for the setting when both the data and model can be partitioned across sites in a network i.e. all the sites observe the same set of instances but have access to only a subset of features. Each local site constructs a model by feed forward learning and exchanges losses with a randomly chosen neighbor. The losses are averaged and used in

back propagation at the local sites. Empirical results on several real world datasets reveal that the consensus algorithm has performance comparable to the centralized counterpart. In future, we will explore the possibility of extending the algorithm to other kinds of neural networks including convolutional, recurrent and LSTM algorithms and provide an intuition of theoretical proof of convergence of the proposed algorithm.

References

M. K. Abdel-Aziz, C. F. Liu, S. Samarakoon, M. Bennis, and W. Saad. Ultra-reliable low-latency vehicular networks: Taming the age of information tail. *IEEE Global Communications Conference*, pages 1–7, 2018.

D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pages 1709–1720. 2017.

N. Batra, J. Kelly, O. Parson, H. Dutta, W. Knottenbelt, A. Rogers, A. Singh, and M. Srivastava. Nilmtk: An open source toolkit for non-intrusive load monitoring. In *Proceedings of the 5th International Conference on Future Energy Systems*, page 265276, 2014.

R. Bekkerman, M. Bilenko, and J. Langford. *Scaling Up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press, New York, NY, USA, 2011.

M.l Blot, D. Picard, N. Thome, and M. Cord. Distributed Optimization for Deep Learning with Gossip Exchange. *Neurocomputing*, 330:287–296, 2019.

Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recogn.*, 30(7):11451159, July 1997. ISSN 0031-3203. doi: 10.1016/S0031-3203(96)00142-2. URL https://doi.org/10.1016/S0031-3203(96)00142-2.

Leo Breiman. Random forests. *Mach. Learn.*, 45(1):532, October 2001.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

J. Dean, G. Corrado, R. Monga, C. Kai, M. Devin, M. Mao, R. Marc, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25*, pages 1223–1231. 2012.

A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. *ACM Symposium on Principles of Distributed Computing*, pages 1–12, 1987.

O. Gupta and R. Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1 – 8, 2018a. ISSN 1084-8045.

O. Gupta and R. Raskar. Distributed learning of deep neural network over multiple agents. *J. Netw. Comput. Appl.*, 116:1–8, 2018b.

I. Guyon, S. Gunn, A. B. Hur, and G. Dror. Result analysis of the nips 2003 feature selection challenge. In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, NIPS'04, pages 545–552, 2004.

J.A. Hanley and Barbara Mcneil. A method of comparing the areas under receiver operating characteristic curves derived from the same cases. *Radiology*, 148:839–43, 10 1983. doi: 10.1148/radiology.148.3.6878708.

R. Huerta, T. Mosqueiro, J. Fonollosa, F. Rulkov, N., and I. Rodriguez-Lujan. Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. *Chemometrics and Intelligent Laboratory Systems*, 157:169176, Oct 2016a.

R. Huerta, T. Mosqueiro, J. Fonollosa, N. F. Rulkov, and I. Rodríguez-Luján. Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. *Chemometrics and Intelligent Laboratory Systems*, 157:169–176, 2016b.

Z. Jiang, A. Balu, C. Hegde, and S. Sarkar. Collaborative deep learning in fixed topology networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5906–5916, 2017.

Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *SIGARCH Comput. Archit. News*, 45(1):615–629, April 2017.

D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. *IEEE Symposium on Foundations of Computer Science*, pages 482–491, 2003.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

C. Lin, T. Wang, K. Chen, B. Lee, and J. Kuo. Distributed deep neural network deployment for smart devices from the edge to the cloud. In *Proceedings of the ACM MobiHoc Workshop on Pervasive Systems in the IoT Era*, PERSIST-IoT '19, pages 43–48, 2019.

C. S. Lin, Y. Zhang, C. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars. The architectural implications of autonomous driving: Constraints and acceleration. *SIGPLAN Not.*, 53 (2):751766, March 2018.

R. McDonald, K. Hall, and G. Mann. Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 456–464, 2010.

B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS*, pages 1273–1282, 2017.

A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pages 99–100, September 2009.

J. Park, S. Samarakoon, M. Bennis, and M. Debbah. Wireless network intelligence at the edge. *Proceedings of the IEEE*, 107(11):2204–2239, 2019.

F. Provost and T. Fawcett. *Data Science for Business*. OReilly, New York, NY, USA, 2013.

F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. 1-bit stochastic gradient descent and application to data-parallel distributed training of speech dnns. In *Interspeech 2014*, September 2014.

D. P. Sutton, M. C. Carlisle, T. A. Sarmiento, and L. C. Baird. Partitioned neural networks. In *Proceedings of the 2009 International Joint Conference on Neural Networks*, IJCNN'09, pages 2870–2875, 2009.

S. Teerapittayanon, B. McDanel, and H. T. Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA*, pages 328–339, 2017.

P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *ArXiv*, abs/1812.00564, 2018.

X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 22(2):869–904, 2020.

W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems 30*, pages 1509–1519. 2017.

W. Zhang, S. Gupta, X. Lian, and J. Liu. Staleness-aware async-sgd for distributed deep learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 2350–2356, 2016.

S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8): 1738–1762, 2019.

Shlomo Zilberstein. *Operational Rationality through Compilation of Anytime Algorithms*. PhD thesis, Computer Science Division, University of California Berkeley, 1993.

Shlomo Zilberstein and S. J. Russell. Anytime sensing, planning and action: A practical model for robot control. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1402–1407, Chambery, France, 1993.