# DISTRIBUTED LARGE NEURAL NETWORK WITH CENTRALIZED EQUIVALENCE

*Xinyue Liang, Alireza M. Javid, Mikael Skoglund, Saikat Chatterjee*

Department of Information Science and Engineering
School of Electrical Engineering, KTH Royal Institute of Technology, Sweden
xinyuel@kth.se, almj@kth.se, skoglund@kth.se, sach@kth.se

## ABSTRACT

In this article, we develop a distributed algorithm for learning a large neural network that is deep and wide. We consider a scenario where the training dataset is not available in a single processing node, but distributed among several nodes. We show that a recently proposed large neural network architecture called progressive learning network (PLN) can be trained in a distributed setup with centralized equivalence. That means we would get the same result if the data be available in a single node. Using a distributed convex optimization method called alternating-direction-method-of-multipliers (ADMM), we perform training of PLN in the distributed setup.

***Index Terms***— Distributed learning, neural networks, data parallelism, convex optimization.

## 1. INTRODUCTION

Artificial neural networks have seen a great revival in recent years. Many efforts have been carried out to improve the performance capabilities of the primary neural network structures, for example, recurrent neural networks [1], residual networks [2], deep networks [3], extreme learning machines [4, 5], etc. These example neural networks are shown to provide high quality performances for many applications, for example in speech recognition, object recognition, image representation, biological analysis, and more [6–12]. In this article, we consider a scenario where training data is not available in a single node (here a node can be a processor / memory), but distributed over several nodes. This is the distributed scenario we are dealing with which can be motivated for two main reasons: (a) the training data is very large and can not be accommodated in a single node, (b) training data can not be placed in a single node or shared between the nodes due to privacy issues. This distributed scenario relates to the term 'data parallelism' in the neural network community [13]. For this scenario, our main objective in this article is to design a distributed training algorithm. The distributed algorithm ensures training of a large neural network that has the same structure and parameters as if it were trained in a centralized manner. Here the term 'centralized manner' corresponds to the case where all the training data could be accessed in a single node.

Relevant existing work on distributed learning of neural networks includes [14–17]. All these prior works use model parallelism or data parallelism. In model parallelism, nodes are responsible for learning different parts of a single network, for example, each layer in the neural network may be handled in each node. On the other hand, in data parallelism, the data is distributed over nodes, as in the case we are interested in, and all nodes try to arrive at the same model where learned models over nodes are somehow combined. An important aspect is that none of the mentioned works in data parallelism can provide a distributed training algorithm that ensures equivalence to a centralized solution.

The main bottleneck is to implement an efficient distributed solution of the centralized training algorithm. The most prominent approach to learn a neural network is back propagation using gradient search. It is non-trivial to design a distributed back propagation algorithm that provides a centralized solution. We find a recent endeavor where a distributed extreme learning machine [18] is developed which can arrive at the same solution as that of the centralized approach. This was possible as extreme learning machine only learns parameters using least-squares, and least-squares can be efficiently solved using a distributed optimization algorithm called alternating-direction-method-of-multipliers (ADMM) [19]. At this point, we mention that an extreme learning machine is typically a shallow network comprised of a few wide layers. In [20], a progressive learning network (PLN) was developed where parameters of the layers are learned using constrained least-squares. PLNs are large in the sense that they can have many hidden neurons and many layers. Therefore, to design a large neural network in the distributed scenario, we train the PLN in a distributed manner where all constrained least-squares optimization problems are solved using ADMM. The use of ADMM in distributed training results in the same solution as the centralized scenario.

### 1.1. Distributed setup with centralized equivalence

In a supervised learning problem, let $\mathbf{d} = (\mathbf{x}, \mathbf{t})$ be a pair-wise form of input data vector $\mathbf{x}$ that we observe and target vector $\mathbf{t}$ that we wish to infer. We assume $\mathbf{x} \in \mathbb{R}^P$ and $\mathbf{t} \in \mathbb{R}^Q$. A neural network is an inference function that provides an output $\tilde{\mathbf{t}} = \mathbf{f}(\mathbf{x}, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ contains the parameters of the neural network. Let us use $(\mathbf{x}^{(j)}, \mathbf{t}^{(j)})$ to denote the $j$'th data-and-target pair and assume that we have $J$ such training samples as a training dataset $\mathcal{D} = \{(\mathbf{x}^{(j)}, \mathbf{t}^{(j)})\}_{j=1}^{J}$. Note that the cardinality of $\mathcal{D}$ is $J$. Let us use a cost function

$$C(\boldsymbol{\theta}) = \sum_{(\mathbf{x}^{(j)}, \mathbf{t}^{(j)}) \in \mathcal{D}} \|\mathbf{t}^{(j)} - \mathbf{f}(\mathbf{x}^{(j)}, \boldsymbol{\theta})\|_2^2. \tag{1}$$

For a centralized scenario, we have all the $J$ samples in a single node. In the training phase, we learn optimal parameters by minimizing the cost in a regularized manner, as follows

$$\boldsymbol{\theta}_c^\star = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, C(\boldsymbol{\theta}) \text{ s.t. } \|\boldsymbol{\theta}\|_2^2 \leq \epsilon, \tag{2}$$

where 's.t.' is the abbreviation of 'such that'. The above problem is non-convex for a general neural network and hence, non-trivial.

In a distributed scenario, we have $M$ processing nodes and we assume that the training dataset $\mathcal{D}$ is divided into $M$ datasets. We denote the training dataset at the $m$'th node by $\mathcal{D}_m$ such that $\cup_m \mathcal{D}_m =$

$\mathcal{D}$. Let us denote the cost at the $m$'th node by

$$C(\boldsymbol{\theta}_m) = \sum_{(\mathbf{x}^{(j)},\mathbf{t}^{(j)})\in\mathcal{D}_m} \|\mathbf{t}^{(j)} - \mathbf{f}(\mathbf{x}^{(j)},\boldsymbol{\theta}_m)\|_2^2 \tag{3}$$

where $\boldsymbol{\theta}_m$ denotes the parameters learned at the $m$'th node. Our interest is to learn the parameters in a distributed manner as follows

$$\boldsymbol{\theta}_d^\star = \underset{\boldsymbol{\theta},\boldsymbol{\theta}_m}{\operatorname{argmin}} \sum_m C(\boldsymbol{\theta}_m) \text{ s.t. } \forall m,\, \boldsymbol{\theta}_m = \boldsymbol{\theta},\, \|\boldsymbol{\theta}\|_2^2 \le \epsilon. \tag{4}$$

The constraint $\forall m, \boldsymbol{\theta}_m = \boldsymbol{\theta}$ enforces the same solution for all nodes. For the PLN, we can achieve $\boldsymbol{\theta}_c^\star = \boldsymbol{\theta}_d^\star$ using ADMM under some technical conditions. This is the centralized equivalence. In the next section, we briefly describe the PLN.

## 2. PROGRESSIVE LEARNING NETWORK

A standard architecture for artificial neural networks (ANNs) is comprised of several layers where signal transformation flows from the input side to the output side, that is, in one direction. This is known as a feed-forward neural network. Each layer of an ANN is comprised of a linear transform (LT) of an input vector, followed by a non-linear transform (NLT) to generate an output vector. The output vector of a layer is then used as an input vector to the next layer. The linear transform is represented by a weight matrix, whereas the non-linear transform of the input vector is typically realized by a scalar-wise non-linear transform, known as activation function. A popular activation function is the rectifier linear unit (ReLU) that was used to develop the PLN. Based on progression property of ReLU (explained in [20]), PLN uses a layer-wise design principle. A new layer is added to an existing optimized network, and each new layer is then learned and optimized at a time. For learning of each layer, we have a convex optimization problem to solve. This is the main reason that we can use a distributed convex optimization algorithm ADMM to develop a distributed PLN.

The PLN architecture is shown in Figure 1, where each layer is comprised of a linear input transform and a non-linear transform. For simplicity, we assume that the number of hidden neurons in all layers are the same. We denote the number of hidden neurons in each and every layer by $n$. We use $\mathbf{W}_l$ to denote the linear transform related to the $l$'th layer. If the PLN has $L$ layers, then $\mathbf{W}_1$ is $n \times P$ size, and all the other weight matrices $\mathbf{W}_l|_{l=2}^{L-1}$ are of $n \times n$ size. The weight matrices have the following structure

$$\mathbf{W}_l = \begin{cases} \begin{bmatrix} \mathbf{V}_Q \mathbf{W}_{ls}^\star \\ \mathbf{R}_1 \end{bmatrix} & \text{if } l = 1 \\ \begin{bmatrix} \mathbf{V}_Q \mathbf{O}_{l-1}^\star \\ \mathbf{R}_l \end{bmatrix} & \text{for } l = 2, 3, \ldots, L, \end{cases} \tag{5}$$

where $\mathbf{W}_{ls}^\star \in \mathbb{R}^{Q\times P}$ denotes the optimum linear transform for a target approximation linear system and $\mathbf{O}_l^\star \in \mathbb{R}^{Q\times n}$ denotes the optimum output matrix for $l$'th layer in the PLN; further $\mathbf{V}_Q = [\mathbf{I}_Q \ -\mathbf{I}_Q]^T$ is a known matrix of $2Q \times Q$ size and $\mathbf{R}_l$ are instances of random matrices. The matrix $\mathbf{R}_1$ is $(n-2Q)\times P$ size and other $\mathbf{R}_l$ matrices are $(n-2Q)\times n$ size. Let us construct data and target matrices as follows

$$\begin{aligned} \mathbf{X} &= \left[\mathbf{x}^{(1)}\, \mathbf{x}^{(2)}\, \ldots\, \mathbf{x}^{(J)}\right] \in \mathbb{R}^{P\times J}, \\ \mathbf{T} &= \left[\mathbf{t}^{(1)}\, \mathbf{t}^{(2)}\, \ldots\, \mathbf{t}^{(J)}\right] \in \mathbb{R}^{Q\times J}. \end{aligned} \tag{6}$$

The optimum linear transform $\mathbf{W}_{ls}^\star$ is found by solving a regularized least-squares problem (a convex optimization problem), as follows

$$\mathbf{W}_{ls}^\star = \underset{\mathbf{W}_{ls}}{\operatorname{argmin}} \|\mathbf{T} - \mathbf{W}_{ls}\mathbf{X}\|_F^2 + \lambda\|\mathbf{W}_{ls}\|_F^2, \tag{7}$$

where $\lambda$ is a parameter to tune. Let us denote $n$ ReLU functions together by a vector non-linear function $\mathbf{g}$. Then, in the first layer of PLN, the signal after non-linear transformation is $\mathbf{y}_1 = \mathbf{g}(\mathbf{W}_1\mathbf{x})$. Similarly, for the $l$'th layer of PLN, the signal after non-linear transformation is $\mathbf{y}_l = \mathbf{g}(\mathbf{W}_l\mathbf{y}_{l-1})$. We use the notation $\mathbf{y}_l^{(j)}$ to denote the signal $\mathbf{y}_l$ for the $j$'th input data $\mathbf{x}^{(j)}$. Let us construct the signal matrix at the $l$'th layer

$$\mathbf{Y}_l = \left[\mathbf{y}^{(1)}\, \mathbf{y}^{(2)}\, \ldots\, \mathbf{y}^{(J)}\right] \in \mathbb{R}^{n\times J}. \tag{8}$$

For the $l$'th layer, we find an optimal output matrix for target approximation by using the following convex optimization problem

$$\mathbf{O}_l^\star = \underset{\mathbf{O}_l}{\operatorname{argmin}} \|\mathbf{T} - \mathbf{O}_l\mathbf{Y}_l\|_F^2 \text{ s.t. } \|\mathbf{O}_l\|_F^2 \le \alpha\|\mathbf{U}_Q\|_F^2, \tag{9}$$

where $\alpha \ge 1$ is a parameter we choose and $\mathbf{U}_Q = [\mathbf{I}_Q \ -\mathbf{I}_Q] \in \mathbb{R}^{Q\times 2Q}$ is a known matrix. In PLN, we construct layers one-by-one and learn the parameter $\mathbf{O}_l^\star$ using the above convex optimization formulation. A PLN with $l$ layers is built on an optimized PLN with $(l-1)$ layers. We start with a single-layer PLN, then build a two-layer PLN, and then proceed further successively in a layer-wise fashion to develop a multi-layer PLN. For the training dataset, let us denote the optimum cost for the $l$'th layer of PLN by $C_l^\star = \|\mathbf{T} - \mathbf{O}_l^\star\mathbf{Y}_l\|_F^2$. Then, by construction of PLN and using the progression property of the ReLU function, we can show that the cost is non-increasing with growing number of layers; analytically it means that PLN satisfies $C_l^\star \le C_{l-1}^\star$. In ReLU based PLN, the parameters to learn are $\{\mathbf{W}_l\}$ matrices. In effect, the parameters to learn are $\{\mathbf{W}_{ls}^\star, \{\mathbf{O}_l^\star\}\}$, and the parameters to tune are $\lambda, \alpha$. The $\mathbf{R}_l$ matrices are random matrix instances, and fixed once chosen.

## 3. DISTRIBUTED LEARNING OF PLN

In a distributed setup of $M$ processing nodes, we assume that the $m$'th node has $J_m$ training samples. Let us write the data matrix $\mathbf{X}$ and the target matrix $\mathbf{T}$ as below

$$\begin{aligned} \mathbf{X} &= [\mathbf{X}_1\, \mathbf{X}_2\, \ldots\, \mathbf{X}_m\, \ldots\, \mathbf{X}_M], \\ \mathbf{T} &= [\mathbf{T}_1\, \mathbf{T}_2\, \ldots\, \mathbf{T}_m\, \ldots\, \mathbf{T}_M], \end{aligned} \tag{10}$$

where $\mathbf{X}_m \in \mathbb{R}^{P\times J_m}$ is the input data matrix in the $m$'th node and $\mathbf{T}_m \in \mathbb{R}^{Q\times J_m}$ is the target data matrix accordingly. We assume that $\mathbf{X}_m$ and $\mathbf{T}_m$ together form the dataset $\mathcal{D}_m$. For learning the parameters of PLN in the distributed scenario, we first focus on learning $\mathbf{W}_{ls}^\star$ as per (7). We recast the learning problem in the following optimization problem

$$\begin{aligned} \min_{\mathbf{W}_m,\mathbf{Z}} \textstyle\sum_{m=1}^M \|\mathbf{T}_m - \mathbf{W}_m\mathbf{X}_m\|_F^2 + \lambda\|\mathbf{Z}\|_F^2 \\ \text{s.t. } \forall m, \mathbf{W}_m = \mathbf{Z}. \end{aligned} \tag{11}$$

where $\mathbf{Z}$ is an auxiliary variable. We solve the above optimization problem using ADMM. We break (11) into the following three parts that can be executed in each of the $M$ nodes using relevant information exchanged between nodes:

$$\begin{aligned} \mathbf{W}_m^\star &= \underset{\mathbf{W}}{\operatorname{argmin}} \|\mathbf{T}_m - \mathbf{W}\mathbf{X}_m\|_F^2 + \lambda\|\mathbf{Z}_m\|_F^2 + \tfrac{1}{\rho}\|\mathbf{W} - \mathbf{Z}_m + \boldsymbol{\Lambda}_m\|_F^2, \\ \mathbf{Z}_m^\star &= \underset{\mathbf{Z}}{\operatorname{argmin}}\ \lambda\|\mathbf{Z}\|_F^2 + \tfrac{1}{\rho}\textstyle\sum_{m=1}^M \|\mathbf{W}_m^\star - \mathbf{Z} + \boldsymbol{\Lambda}_m\|_F^2, \\ \boldsymbol{\Lambda}_m &= \boldsymbol{\Lambda}_m + \mathbf{W}_m^\star - \mathbf{Z}_m^\star. \end{aligned}$$

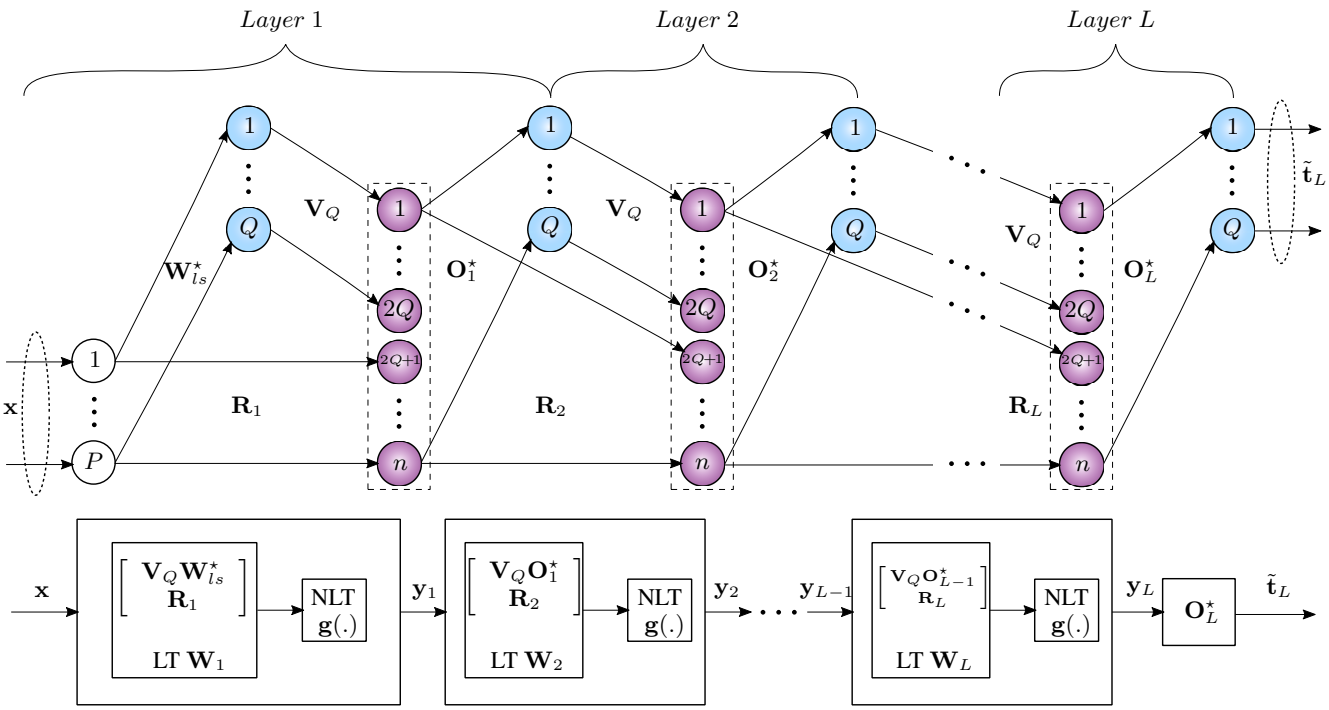


**Fig. 1**. The architecture of a multi-layer PLN with $L$ layers. 'LT' stands for linear transform and 'NLT' stands for non-linear transform.

Note that the minimization problems have closed form solutions, and therefore, the ADMM iterations are :

$$\begin{cases} \mathbf{W}_m^{k+1} = \left(\mathbf{T}_m\mathbf{X}_m^T + \frac{1}{\rho}(\mathbf{Z}_m^k - \mathbf{\Lambda}_m^k)\right)\cdot(\mathbf{X}_m\mathbf{X}_m^T + \frac{1}{\rho}\mathbf{I})^{-1}, \\ \mathbf{Z}_m^{k+1} = \frac{1}{\rho\lambda+M}\sum_{n=1}^{M}(\mathbf{W}_n^{k+1} + \mathbf{\Lambda}_n^k), \\ \mathbf{\Lambda}_m^{k+1} = \mathbf{\Lambda}_m^k + \mathbf{W}_m^{k+1} - \mathbf{Z}_m^{k+1}, \end{cases}$$

where $k$ denotes the iteration number in ADMM. Note that all $\mathbf{W}_m$ matrices need to be first updated and shared between all nodes (communicated to each other) in order to move to the second part of the ADMM iteration which is updating $\mathbf{Z}_m$.

Next we focus on learning $\mathbf{O}_l^\star$ in (9). Let us use a notation $\epsilon_o \triangleq \alpha\|\mathbf{U}_Q\|_F^2$. For the $m$'th node, we denote the signal matrix at the $l$'th layer of the PLN as $\mathbf{Y}_{l,m}$. Note that $\mathbf{Y}_{l,m}$ is generated by feeding the input data matrix $\mathbf{X}_m$ to the corresponding PLN. The column vectors of $\mathbf{Y}_{l,m}$ are the output vectors at the $l$'th layer for the input data vectors in $\mathbf{X}_m$. We recast the learning problem (9) as the following optimization problem:

$$\min_{\mathbf{O}_{l,m},\mathbf{Z}} \sum_{m=1}^{M}\|\mathbf{T}_m - \mathbf{O}_{l,m}\mathbf{Y}_{l,m}\|_F^2 \text{ s.t. } \|\mathbf{Z}\|_F^2 \le \epsilon_o, \quad \forall m, \mathbf{O}_{l,m} = \mathbf{Z}. \tag{12}$$

Then, we break it into three parts:

$$\begin{aligned} \mathbf{O}_{l,m}^\star &= \underset{\mathbf{O}}{\text{argmin}}\ \|\mathbf{T}_m - \mathbf{O}\mathbf{Y}_{l,m}\|_F^2 + \tfrac{1}{\mu}\|\mathbf{O} - \mathbf{Z}_m + \mathbf{\Lambda}_m\|_F^2, \\ \mathbf{Z}_m^\star &= \underset{\mathbf{Z}}{\text{argmin}}\ \sum_{m=1}^{M}\|\mathbf{O}_{l,m}^\star - \mathbf{Z} + \mathbf{\Lambda}_m\|_F^2 \text{ s.t. } \|\mathbf{Z}\|_F^2 \le \epsilon_o, \\ \mathbf{\Lambda}_m &= \mathbf{\Lambda}_m + \mathbf{O}_{l,m}^\star - \mathbf{Z}_m^\star. \end{aligned}$$

The ADMM iterations would be:

$$\begin{cases} \mathbf{O}_{l,m}^{k+1} = \left(\mathbf{T}_m\mathbf{Y}_{l,m}^T + \frac{1}{\mu}(\mathbf{Z}_m^k - \mathbf{\Lambda}_m^k)\right)\cdot(\mathbf{Y}_{l,m}\mathbf{Y}_{l,m}^T + \frac{1}{\mu}\mathbf{I})^{-1}, \\ \mathbf{Z}_m^{k+1} = \mathcal{P}_{\epsilon_o}(\frac{1}{M}\sum_{n=1}^{M}(\mathbf{O}_{l,n}^{k+1} + \mathbf{\Lambda}_n^k)), \\ \mathbf{\Lambda}_m^{k+1} = \mathbf{\Lambda}_m^k + \mathbf{O}_{l,m}^{k+1} - \mathbf{Z}_m^{k+1}. \end{cases}$$

Here, $\mathcal{P}_{\epsilon_o}$ performs projection onto the space of matrices with $\|.\|_F^2 \le \epsilon_o$, that is:

$$\mathcal{P}_{\epsilon_o}(\mathbf{Z}) = \begin{cases} \mathbf{Z}\cdot(\frac{\epsilon_o}{\|\mathbf{Z}\|_F}) & : \|\mathbf{Z}\|_F > \epsilon_o \\ \mathbf{Z} & : \text{otherwise.} \end{cases} \tag{13}$$

As we are learning PLN in a layer-by-layer fashion, the use of ADMM enforces a full synchronization in learning. For learning the $\mathbf{O}_l^\star$ matrix for each layer, we allow ADMM to iterate sufficiently enough to converge and expect to provide the same parameters across all nodes. Here, 'sufficiently enough' is a qualitative term, and in practice, we use a maximum number of iterations for ADMM, say $k_{max} = 100$. While ADMM theoretically should converge to a single solution, in practice there might be some difference between the learned parameters across nodes due to the finite number of iterations. In our experiments, we show that the practical use of ADMM does not result in a tangible performance difference between the centralized and distributed PLN. For the distributed PLN, the parameters $\lambda, \mu, \rho, \epsilon_o$ and $\mathbf{R}_l$ matrices are assumed to be fixed a-priori. These parameters are shared among all nodes before ADMM starts iterating. Then, we develop the PLN layer-by-layer until the cost converges. We can stop learning of PLN when we find that adding a new layer does not result in a significant reduction in the objective cost. However, we use a fixed number of $L$ layers for simplicity, which is known as a-priori.

## 4. EXPERIMENTAL RESULTS

We test the performance of the distributed PLN for classification tasks using simulations. The datasets that we use are briefly mentioned in Table 1. We use the $Q$-dimensional target vector $\mathbf{t}$ in a classification task as a discrete variable with indexed representation of 1-out-of-$Q$-classes. A target variable (vector) instance has only

**Table 1**. Databases for multi-class classification

| Database | # of train data | # of test data | Input dimension ($P$) | # of classes ($Q$) |
|---|---|---|---|---|
| Vowel | 528 | 462 | 10 | 11 |
| Extended YaleB | 1600 | 800 | 504 | 38 |
| AR | 1800 | 800 | 540 | 100 |
| Satimage | 4435 | 2000 | 36 | 6 |
| Scene15 | 3000 | 1400 | 3000 | 15 |
| Caltech101 | 6000 | 3000 | 3000 | 102 |
| Letter | 13333 | 6667 | 16 | 26 |
| NORB | 24300 | 24300 | 2048 | 5 |
| MNIST | 60000 | 10000 | 784 | 10 |

**Table 2**. Classification performance comparison between PLN and distributed PLN where $M = 5$, $L = 10$, $n = 2Q + 100$

| Dataset | Centralized PLN | | Distributed PLN | |
|---|---|---|---|---|
| | Test Accuracy | Training Time(s) | Test Accuracy | Training Time(s) |
| Vowel | 57.3±2.64 | 0.1519 | 56.5±2.76 | 0.0534 |
| Extended YaleB | 96.6±0.76 | 0.9724 | 96.8±0.74 | 0.3335 |
| AR | 95.3±0.94 | 3.5289 | 95.6±0.35 | 1.2553 |
| Satimage | 88.1±0.50 | 0.9471 | 87.9±0.29 | 0.2011 |
| Scene15 | 98.9±0.36 | 2.9818 | 98.9±0.28 | 2.0322 |
| Caltech101 | 74.1±0.83 | 15.815 | 73.5±0.71 | 7.3875 |
| Letter | 87.6±0.68 | 5.0599 | 87.5±0.28 | 1.0479 |
| NORB | 83.1±0.37 | 7.4850 | 83.1±0.17 | 2.1555 |
| MNIST | 91.5±0.17 | 12.479 | 91.3±0.23 | 2.5991 |

one scalar component that is 1, and the other scalar components are zero. Then, for the PLN, we fix the number of layers $L = 10$ and the number of hidden neurons $n = 2Q + 100$ for each layer. We fix the number of processing nodes $M = 5$ and then, uniformly divided the training dataset between the nodes. The classification performance results are reported in Table 2. Our interest is to compare the distributed PLN with the centralized PLN. Note that the distributed PLN uses ADMM with finite number of iterations. We keep the maximum number of iterations as 100. While we mentioned that there exists a distributed ELM [18] that provides equivalence to a centralized ELM, we do not discuss ELM in this article. PLN was thoroughly compared with the ELM in [20] and shown to perform better for most cases. In the reported results in Table 2, we also did not endeavor to find the parameters of PLN such that it provides high quality performance, as reported in [20]. We provide the parameters for centralized and distributed PLN in Table 3.

For the (centralized) PLN, it was shown in [20] that addition of layers leads to a non-increasing trend in the objective cost. The objective cost is in (12) as $\sum_{m=1}^{M} \|\mathbf{T}_m - \mathbf{O}^{\star}_{l,m}\mathbf{Y}_{l,m}\|_F^2$ for the $l$'th layer of PLN. Therefore, it is interesting to see how the decreasing trend of the objective cost gets affected by the number of processing nodes for a fixed amount of training data. Keeping the total amount of training data fixed, if we increase the number of nodes, then, each node has access to a fewer number of training samples.

For the Scene15 database, we did experiments to investigate the trend in the objective cost versus layer number for different processing nodes $M$. The results are shown in Figure 2. It can be seen that as we increase the number of nodes, the performance difference between the distributed PLN and centralized PLN are more pronounced when the number of layers are small. All the curves for different $M$

**Table 3**. The corresponding parameters of Table 2

| Dataset | Centralized PLN | | | Distributed PLN | | | |
|---|---|---|---|---|---|---|---|
| | $\lambda$ | $\mu$ | $\alpha$ | $\lambda$ | $\rho$ | $\mu$ | $\alpha$ |
| Vowel | $10^2$ | $10^{-1}$ | 2 | $10^2$ | $10^{-1}$ | $10^{-1}$ | 2 |
| Extended YaleB | $10^4$ | $10^3$ | 2 | $10^4$ | $10^{-5}$ | $10^{-2}$ | 2 |
| AR | $10^5$ | $10^3$ | 2 | $10^5$ | $10^{-4}$ | $10^1$ | 2 |
| Satimage | $10^6$ | $10^5$ | 2 | $10^6$ | $10^{-4}$ | $10^{-2}$ | 2 |
| Scene15 | $10^{-3}$ | $10^4$ | 2 | $10^{-3}$ | $10^2$ | $10^0$ | 2 |
| Caltech101 | $10^0$ | $10^{-2}$ | 2 | $10^0$ | $10^{-1}$ | $10^{-1}$ | 2 |
| Letter | $10^{-5}$ | $10^6$ | 2 | $10^{-5}$ | $10^{-6}$ | $10^{-1}$ | 2 |
| NORB | $10^2$ | $10^{-3}$ | 2 | $10^2$ | $10^{-3}$ | $10^{-3}$ | 2 |
| MNIST | $10^0$ | $10^5$ | 2 | $10^0$ | $10^{-2}$ | $10^{-2}$ | 2 |

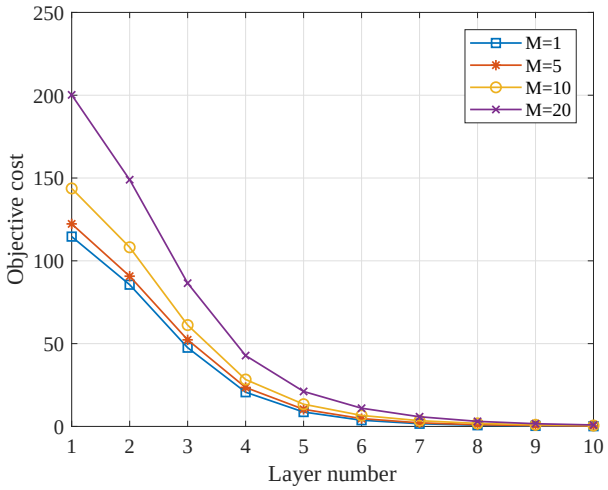


**Fig. 2**. Objective Cost versus layer number, Scene15 database, $k_{max} = 100$

reach to similar performance levels when the number of layers increases. This result is interesting as it shows that increase in the number of layers in a neural network may lead to a robust performance in the distributed scenario. Thus, deep neural networks might have a higher potential in a distributed scenario than a shallow neural network.

Matlab codes of all the experiments described in this paper are available at https://sites.google.com/site/saikatchatt/. The datasets used for the experiments can be found at [21–24] .

## 5. CONCLUSIONS AND DISCUSSIONS

We conclude that a distributed algorithm can be developed for learning a large neural network with multiple layers. To this end, we used ADMM algorithm for optimizing each layer of progressive learning network (PLN) and showed that the learned network is equivalent to that of the centralized scenario under some technical conditions. In this work, we have assumed that all processing nodes have direct access to each other in order to transmit required parameters to realize the distributed algorithm. In other words, we have assumed a complete graph to characterize the communication network topology over nodes. A future direction is to consider learning a distributed neural network over a non-complete graph, where a node only has access to information from few neighboring nodes.

## 6. REFERENCES


[1] Tomas Mikolov, Martin Karafit, Lukas Burget, Jan Cernock, and Sanjeev Khudanpur, "Recurrent neural network based language model," 01 2010.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[4] G. Huang, Q. Zhu, and C. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489 – 501, 2006, Neural Networks.

[5] G. Huang, Q. Zhu, and C. Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks," in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, July 2004, vol. 2, pp. 985–990 vol.2.

[6] N. Hassan, D. A. Ramli, and H. Jaafar, "Deep neural network approach to frog species recognition," in *2017 IEEE 13th International Colloquium on Signal Processing its Applications (CSPA)*, March 2017, pp. 173–178.

[7] A. Khotanzad and J. H. Lu, "Classification of invariant image representations using a neural network," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 6, pp. 1028–1038, Jun 1990.

[8] Y. Tsuzuki, K. Sawada, K. Hashimoto, Y. Nankaku, and K. Tokuda, "Image recognition based on discriminative models using features generated from separable lattice hmms," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2017, pp. 2607–2611.

[9] D. Ellis and N. Morgan, "Size matters: an empirical study of neural network training for large vocabulary continuous speech recognition," in *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, Mar 1999, vol. 2, pp. 1013–1016 vol.2.

[10] J. H. L. Hansen and T. Hasan, "Speaker recognition by machines and humans: A tutorial review," *IEEE Signal Processing Magazine*, vol. 32, no. 6, pp. 74–99, Nov 2015.

[11] A. Stuhlsatz, C. Meyer, F. Eyben, T. Zielke, G. Meier, and B. Schuller, "Deep neural networks for acoustic emotion recognition: Raising the benchmarks," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2011, pp. 5688–5691.

[12] Z. Q. Wang and I. Tashev, "Learning utterance-level representations for speech emotion and age/gender recognition using deep neural networks," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2017, pp. 5150–5154.

[13] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., pp. 1223–1231. Curran Associates, Inc., 2012.

[14] J. Xin, Z. Wang, L. Qu, G. Yu, and Y. Kang, "A-elm: Adaptive distributed extreme learning machine with mapreduce," *Neurocomputing*, vol. 174, no. Part A, pp. 368 – 374, 2016.

[15] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 328–339.

[16] K. Chang, N. Balachandar, C. K Lam, D. Yi, J. M Brown, A. Beers, B. R Rosen, D. L Rubin, and J. Kalpathy-Cramer, "Institutionally Distributed Deep Learning Networks," *ArXiv e-prints*, Sept. 2017.

[17] D. Das, S. Avancha, D. Mudigere, K. Vaidyanathan, S. Sridharan, D. D. Kalamkar, B. Kaul, and P. Dubey, "Distributed deep learning using synchronous stochastic gradient descent," *CoRR*, vol. abs/1602.06709, 2016.

[18] M. Luo, L. Zhang, J. Liu, J. Guo, and Q. Zheng, "Distributed extreme learning machine with alternating direction method of multiplier," *Neurocomputing*, vol. 261, no. Supplement C, pp. 164 – 170, 2017, Advances in Extreme Learning Machines (ELM 2015).

[19] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011.

[20] S. Chatterjee, A. M. Javid, M. Sadeghi, P. P. Mitra, and M. Skoglund, "Progressive Learning for Systematic Design of Large Neural Networks," *ArXiv e-prints*, Oct. 2017.

[21] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml.

[22] Z. Jiang, Z. Lin, and L. S. Davis, "Learning a discriminative dictionary for sparse coding via label consistent k-svd," in *CVPR 2011*, June 2011., pp. 1697–1704. [Online]. Available: https://www.umiacs.umd.edu/ zhuolin/projectlcksvd.html.

[23] Y. LeCun, Fu Jie Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, June 2004, vol. 2, pp. 97–104 [Online]. Available: http://www.cs.nyu.edu/ ylclab/data/norb–v1.0/.

[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/.