Learning without Forgetting for Decentralized Neural Nets with Low Communication Overhead

Xinyue Liang, Alireza M. Javid, Mikael Skoglund, and Saikat Chatterjee
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology, Sweden
{xinyuel, almj, skoglund, sach}@kth.se

***Abstract*—We consider the problem of training a neural net over a decentralized scenario with a low communication overhead. The problem is addressed by adapting a recently proposed incremental learning approach, called 'learning without forgetting'. While an incremental learning approach assumes data availability in a sequence, nodes of the decentralized scenario can not share data between them and there is no master node. Nodes can communicate information about model parameters among neighbors. Communication of model parameters is the key to adapt the 'learning without forgetting' approach to the decentralized scenario. We use random walk based communication to handle a highly limited communication resource.**

***Index Terms*—Decentralized learning, feedforward neural net, learning without forgetting, low communication overhead**

I. Introduction

Decentralized learning has received a high interest in signal processing, machine learning, and data analysis [1]–[4]. Privacy, security and unavailability of all data in a single place are primary reasons for decentralized learning where data are distributed over several places (or nodes). We consider a decentralized scenario where nodes are not allowed to share data and no master node has access to all nodes. Important aspects of decentralized learning are low computational complexity and low communication overhead to efficiently handle large scale data across nodes.

Decentralized learning is part of the area of distributed learning. Many distributed scenarios assume the existence of a master node. Distributed learning of neural networks mainly uses distributed optimization techniques. In the gamut of distributed learning of neural networks, there exists a considerable focus on distributed gradient descent algorithms for solving non-convex optimization problems [5]–[7]. There are concepts and various implementations of model parallelism and data parallelism [8]–[10]. There are few attempts to design a decentralized learning algorithm to achieve the same performance as that of a centralized setup. The main hindrance is that neural networks often result in non-convex optimization problems with respect to their parameters. We had prior attempts to design distributed neural networks with centralized equivalence [11], [12]. Our strategy was based on the structure of a recently proposed neural network that uses layer-wise convex optimization for training [13]. Alternating-method-of-multipliers (ADMM) was used to handle the layer-wise convex optimization in our past attempts. Instead of layer-wise convex optimization, ADMM was successfully shown to be useful for learning parameters of the last layer in a distributed extreme learning machine [14].

For distributed learning, both gradient descent and ADMM provide good performance while the computational complexity suffers from frequent broadcast or peer-to-peer communication with neighbor nodes. In case of a low communication resource availability, there are efforts to use asynchronous computation and communication for information exchange between nodes [15], [16], asynchronous distributed learning enjoys improved communication efficiency but still suffers when communication resource availability is highly limited.

We now digress to the topic of incremental learning. This topic considers improving a learning model using a new dataset by utilizing the model learned from an old dataset [17]. An incremental learning is defined as one that generates on a given stream of training data $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_M$, a sequence of model parameters $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots \boldsymbol{\theta}_M$. The model parameter $\boldsymbol{\theta}_m$ depends on $\boldsymbol{\theta}_{m-1}$ and $\mathcal{D}_m$. Many efforts have been made to prevent the newly learned model from the issue of catastrophic forgetting [18]. There are three main categories for incremental learning. In the first category, some examples from the old dataset need to be available [19]. The second category uses generative models such as generative adversarial networks (GANs) to generate synthetic data related to the old dataset to train and validate the new learning model [20]. The third category does not require the old dataset or any example from the old dataset. In this category, access to the old dataset is not permitted. A prominent example of the third category is 'learning without forgetting' [21]. The method tries to mimic behavior for the old dataset using a new dataset.

Our contribution in this article is to develop a decentralized learning algorithm for training neural networks based on 'learning without forgetting' approach of incremental learning. Neither nodes are allowed to share data, parameters of data, or any data example, nor any master node exists. Model parameters, that means the parameters of the neural networks, such as weight matrices are allowed to share. Besides we consider the scenario with the availability of a highly limited communication resource. That means we can not communicate model parameters frequently among nodes.

II. Learning without forgetting for a decentralized setup

Let us consider a decentralized setup with M nodes. In a decentralized setup, we assume that the m-th node has J_m

Algorithm 1 : Concept of learning without forgetting for decentralized setup (LwFd)

Model update when the m-th node receives parameters from a neighbour node in random walk based transmission
Start with:

1: Training dataset $\mathcal{D}_m$ for the m-th node; $\mathcal{D}_m$ comprises input $\mathbf{X}_m$ and target $\mathbf{T}_m$
2: $\boldsymbol{\theta}_m$ is the initial parameter of neural net at the m-th node

Definition:

1: $\boldsymbol{\theta}_{r,m}$ (Parameter received at the m-th node from a transmitting neighbour node in random walk)
2: $\boldsymbol{\theta}_{u,m}$ (Parameter updated at the m-th node using learning without forgetting (LwF) approach)
3: $\mathcal{L}$ denotes a loss function and $\mathcal{R}$ denotes an appropriate regularization

Training and learning of neural net (NN) parameters:

1: Compute $\mathbf{T}_{r,m} = \mathrm{NN}(\mathbf{X}_m; \boldsymbol{\theta}_{r,m})$ (Compute output at the 'transmitting neighbour node' with local data $\mathbf{X}_m$)
2: Define $\hat{\mathbf{T}}_{u,m} = \mathrm{NN}(\mathbf{X}_m; \boldsymbol{\theta}_{u,m})$ (Output at the m-th node using updated parameter $\boldsymbol{\theta}_{u,m}$) and local data $\mathbf{X}_m$)
3: LwF strategy: $\boldsymbol{\theta}^{\star}_{u,m} = \mathrm{argmin}_{\boldsymbol{\theta}_{u,m}} \left\{ \mathcal{L}(\mathbf{T}_m, \hat{\mathbf{T}}_{u,m}) + \lambda \mathcal{L}(\mathbf{T}_{r,m}, \hat{\mathbf{T}}_{u,m}) + \mathcal{R}(\boldsymbol{\theta}_{u,m}) \right\}$

training samples. Let us write the data matrix $\mathbf{X}$ and the target matrix $\mathbf{T}$ as below

$$\begin{aligned} \mathbf{X} &= [\mathbf{X}_1\, \mathbf{X}_2\, \ldots\, \mathbf{X}_m\, \ldots\, \mathbf{X}_M], \\ \mathbf{T} &= [\mathbf{T}_1\, \mathbf{T}_2\, \ldots\, \mathbf{T}_m\, \ldots\, \mathbf{T}_M], \end{aligned} \tag{1}$$

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 $\mathcal{D}_m$. The total data is $\mathcal{D} = \cup_{m=1}^{M} \mathcal{D}_m$. In the decentralized setup, nodes can not share data and there is no master node. If it is possible to design a learning algorithm for the decentralized setup such that the algorithm provides the same solution for availability of the total data $\mathcal{D}$ in a single place then the decentralized algorithm has a centralized equivalence. Centralized equivalence is a challenging aspect to achieve and often requires suitable optimization problems, sufficient communication resource and tractable information exchange protocols. For example, if the distributed learning algorithm solves a convex optimization problem over a doubly stochastic communication network.

We here consider a highly limited communication scenario and propose a decentralized learning algorithm for a neural network. In addition, the neural network may not have a convex loss function for optimization. The proposed scheme is called 'learning without forgetting for decentralized setup' (LwFd). To represent a highly limited information communication, we assume that a node can only transmit relevant information to a single neighbor node. To this end, we can use a random walk based information transmission strategy, which perfectly fits with the LwFd scheme. In a random walk based strategy, information transmission starts from a node and the node transmits information to a randomly chosen neighbor node. The transmission process repeats until all the nodes are traversed. With random walk based model parameter transmission, our main technical contribution is the algorithmic formalization of the LwFd scheme. The algorithmic formalization of the LwFd concept is proposed in Algorithm 1. Readers are encouraged to compare the proposed LwFd in Algorithm 1 with Figure 3 of [21] where the concept of 'learning without forgetting' (LwF) is mentioned. Note similarities and differences between LwF used for incremental learning in [21] and the proposed LwF used for the decentralized setup (LwFd).

The LwFd scheme in Algorithm 1 is sufficiently general to use for many types of neural networks, for example, convolutional neural network (CNN) [22]. CNNs use non-convex loss function for optimization, typically by stochastic gradient search. To demonstrate the efficiency of LwFd, we use progressive neural network (PLN) [13]. The prime motivation for using PLN is its low computational complexity. The second motivation is our prior experience in developing decentralized PLN with centralized equivalence [11], [12] and hence realizing a direct performance comparison with a centralized setup. Decentralized PLN with centralized equivalence was possible to engineer due to the use of convex optimization based learning in forming PLN structure.

III. Decentralized PLN using LwFd

In this section, we first briefly discuss the progressive learning network (PLN) as preliminaries, and then propose its realization in a decentralized setup using 'learning without forgetting' approach as per Algorithm 1. Note that the PLN is used as an example in the LwFd approach. The LwFd approach could be extended for many other kinds of neural networks including CNN.

A. Preliminaries about PLN

PLN is a feed-forward neural network that uses a layer-wise learning strategy [13]. Weight matrices are set using a judicious combination of convex optimization and random matrix instances. PLN has a low computational advantage. For demonstrating the decentralized solution of PLN using LwFd, we use a fixed size - which means the PLN has a fixed number of layers and a fixed number of nodes per layer. PLN was shown to self-estimate its own size, while we refrain from size estimation in this article. Estimation of size is not our prime concern in this article. Here we continue with the fixed-size PLN to demonstrate the efficiency of the proposed LwFd.

The PLN addresses the optimization of the network weights in a suboptimal manner. Consider the dataset $(\mathbf{x}, \mathbf{t}) \in \mathcal{D}$,

containing data sample $\mathbf{x} \in \mathbb{R}^P$ and target vector $\mathbf{t} \in \mathbb{R}^Q$. Let us define the feature vector of the l-th layer as

$$\mathbf{y}_l = \mathbf{g}(\mathbf{W}_l\, \mathbf{g}(\dots \mathbf{g}(\mathbf{W}_2\, \mathbf{g}(\mathbf{W}_1\, \mathbf{x}))\dots)) \in \mathbb{R}^h. \tag{2}$$

The layer-by-layer sequential learning approach starts with layer number $l = 0$ and then the layers are increased one-by-one until we reach $l = L$. We set $\mathbf{y}_0 = \mathbf{x}$. Let us first assume that we have an l-layer network. The $(l+1)$-layer network will be built on the l-layer network. For designing the $(l+1)$-layer network given the l-layer network, the steps for finding parameter $\mathbf{W}_{l+1}$ at the $(l+1)$-th layer are as follows:

1) For all the samples in the training dataset $\mathcal{D}$, we compute $\mathbf{y}_l^{(j)} = \mathbf{g}(\mathbf{W}_l\, \mathbf{g}(\dots \mathbf{g}(\mathbf{W}_2\, \mathbf{g}(\mathbf{W}_1\, \mathbf{x}^{(j)}))\dots))$.
2) Using the samples $\{\mathbf{y}_l^{(j)}\}_{j=1}^J$, we solve the following convex optimization problem

$$\mathbf{O}_l^\star = \arg\min_{\mathbf{O}_l} \sum_{j=1}^{J} \|\mathbf{t}^{(j)} - \mathbf{O}_l\, \mathbf{y}_l^{(j)}\|^2 \text{ s.t. } \|\mathbf{O}_l\|_F \le \epsilon_l. \tag{3}$$

It is shown that we can choose the regularization parameters $\epsilon_l = \epsilon = (2Q)^{\frac{1}{2}}$, $l = 0, 1, 2, \dots, L$ to avoid cross-validation. Note that $\mathbf{O}_0$ is $Q \times P$-dimensional, and every $\mathbf{O}_l$ for $l = 1, 2, \dots, L$ is $Q \times h$-dimensional.
3) We form the weight matrix for the $(l+1)$'th layer

$$\mathbf{W}_{l+1} = \begin{bmatrix} \mathbf{V}_Q \mathbf{O}_l^\star \\ \mathbf{R}_{l+1} \end{bmatrix}, \tag{4}$$

where $\mathbf{V}_Q = [\mathbf{I}_Q \ -\mathbf{I}_Q]^T$ is a fixed matrix of dimension $2Q \times Q$, $\mathbf{O}_l^\star$ is learned by convex optimization (3), and $\mathbf{R}_{l+1}$ is an instance of random matrix. The matrix $\mathbf{R}_0$ is $(h-2Q) \times P$-dimensional, and every $\mathbf{R}_l$ for $l = 1, 2, \dots, L$ is $(h-2Q) \times h$-dimensional. Note that we only learn $\mathbf{O}_l^\star$ to form $\mathbf{W}_l$. We do not learn $\mathbf{R}_l$ and it can be pre-fixed before training of PLN. Deciding the weight matrix, the $(l+1)$-layer network is $\mathbf{y}_{l+1} = \mathbf{g}(\mathbf{W}_l\, \mathbf{g}(\dots \mathbf{g}(\mathbf{W}_2\, \mathbf{g}(\mathbf{W}_1\, \mathbf{x}))\dots)) = \mathbf{g}(\mathbf{W}_l \mathbf{y}_l)$.

It is shown in [13] that the three steps mentioned above guarantees monotonically decreasing cost $\sum_j \|\mathbf{t}^{(j)} - \mathbf{O}_l\, \mathbf{y}_l^{(j)}\|^2$ with increasing the layer number l. This property is the key to design a low-complexity architecture as we continue to add new layers one-by-one and set the weight matrix in each layer using (4). It is shown in [13] that the use of gradient descent-based methods for further optimization of parameters in PLN could not provide a reasonable performance improvement.

B. Decentralized PLN using learning without forgetting

Let us consider the decentralized setup of M nodes. The m-th node has J_m training samples, that means $|\mathcal{D}_m| = J_m$. Let us introduce the observation data matrix $\mathbf{X}_m \in \mathbb{R}^{P \times J_m}$ and the target data matrix $\mathbf{T}_m \in \mathbb{R}^{Q \times J_m}$ using the dataset $\mathcal{D}_m$. Here $\mathbf{X}_m$ is formed by the $\mathbf{x}^{(j)} \in \mathcal{D}_m$ and $\mathbf{T}_m$ is formed by the corresponding $\mathbf{t}^{(j)} \in \mathcal{D}_m$.

If no communication happens between the neighboring nodes, the optimal weight parameters at the l-th layer of node m is obtained by solving

$$\mathbf{O}_{l,m}^\star = \min_{\mathbf{O}_{l,m}} \|\mathbf{T}_m - \mathbf{O}_{l,m} \mathbf{Y}_{l,m}\|_F^2, \text{ s.t. } \|\mathbf{O}_{l,m}\|_F \le \epsilon, \tag{5}$$

where $\mathbf{Y}_{l,m}$ is the matrix of feature vectors in equation (2) at the l-th layer of PLN. To solve this optimization problem using ADMM, auxiliary variable $\mathbf{Z}_m$ is introduced to each processing node. The original problem can be rewritten as

$$\begin{aligned} \mathbf{O}_{l,m}^\star = \min_{\mathbf{O}_{l,m}} &\|\mathbf{T}_m - \mathbf{O}_{l,m} \mathbf{Y}_{l,m}\|_F^2, \\ &\text{s.t. } \mathbf{O}_{l,m} = \mathbf{Z}, \|\mathbf{O}_{l,m}\|_F \le \epsilon. \end{aligned} \tag{6}$$

After solving problem (6) for every layer, every node obtains $\{\mathbf{O}_{l,m}^\star\}|_{l=0}^{l=L}$, which is the optimal model parameters for learning the PLN based on the local data. To improve the PLN learned on m-th node, we apply learning without forgetting (LwF) [21] to design a new decentralized learning algorithm using random walk based communication. The random walk starts from a random processing node, if m-th node receives the model parameter set $\{\mathbf{O}_{l,r,m}^\star\}|_{l=0}^{l=L}$ from a neighboring node, LwF is activated at m-th node. Then we reformulate the problem (6) adapting LwF [21] as follows

$$\begin{aligned} \mathbf{O}_{l,u,m}^\star = \min_{\mathbf{O}_{l,u,m}} &\|\mathbf{T}_m - \mathbf{O}_{l,u,m} \mathbf{Y}_{l,m}\|_F^2 \\ &+ \lambda \|\mathbf{O}_{l,r,m}^\star \mathbf{Y}_{l,m} - \mathbf{O}_{l,u,m} \mathbf{Y}_{l,m}\|_F^2, \\ \text{s.t. } &\mathbf{O}_{l,u,m} = \mathbf{Z}, \ \|\mathbf{O}_{l,u,m}\|_F \le \epsilon, \end{aligned} \tag{7}$$

where $\mathbf{O}_{l,r,m}^\star \mathbf{Y}_{l,m}$ is the new generalized target based on the knowledge transmitted from a neighboring node to node m, and λ is a knowledge balance factor. A larger λ favors the knowledge received from the neighboring node, while setting $\lambda = 0$, results in the local optimization in (6).

Using ADMM to solve (7), we break the problem into three parts, and solve them iteratively

$$\begin{aligned} \mathbf{O}_{l,u,m}^{k+1} &= \big(\mathbf{T}_m \mathbf{Y}_{l,m}^T + \lambda \mathbf{O}_{l,r,m}^\star \mathbf{Y}_{l,m} \mathbf{Y}_{l,m}^T \\ &\quad + \tfrac{1}{\mu_l}(\mathbf{Z}^k - \mathbf{\Lambda}^k)\big) \cdot ((1+\lambda)\mathbf{Y}_{l,m} \mathbf{Y}_{l,m}^T + \tfrac{1}{\mu_l}\mathbf{I})^{-1}, \\ \mathbf{Z}^{k+1} &= \mathcal{P}_\epsilon(\mathbf{O}_{l,u,m}^{k+1} + \mathbf{\Lambda}^k), \\ \mathbf{\Lambda}^{k+1} &= \mathbf{\Lambda}^k + \mathbf{O}_{l,u,m}^{k+1} - \mathbf{Z}^{k+1}. \end{aligned} \tag{8}$$

Here, $\mathcal{P}_\epsilon$ is the projection onto the space of matrices with $\|\cdot\|_F \le \epsilon$ to avoid overfitting, which is

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

The above three ADMM updates are executed when the processing node m receives model parameters from neighboring node and activates LwF. This implies that the random walk based communication only happens once in one random walk iteration, and only one node updates its model parameters using the received knowledge. When m-th node finishes updating, it transmits its model parameters to one of its neighbors randomly, and the random walk iteration proceeds. We show the algorithm for LwF for decentralized PLN (LwF-dPLN) in Algorithm 2. Note that Algorithm 2 is an implementation of Algorithm 1 in the context of designing a decentralized PLN. The parameters $\{\mathbf{O}_{l,r,m}^\star\}_{l=0}^{l=L}$ and $\{\mathbf{O}_{l,u,m}^\star\}_{l=0}^{l=L}$ in Algorithm 2 correspond to $\boldsymbol{\theta}_{r,m}$ and $\boldsymbol{\theta}_{u,m}^\star$ in Algorithm 1, respectively.

Algorithm 2 : Algorithm for LwF for decentralized PLN (LwF-dPLN)

Input:

1: Training dataset $\mathcal{D}_m$ for the m-th node
2: Parameters to set manually: L, μ_l, λ
3: Random matrices $\{\mathbf{R}_l\}_{l=1}^{L}$ are shared between all nodes
4: Local learning model parameters $\{\mathbf{O}^{\star}_{l,m}\}|_{l=0}^{l=L}$ learned from local training dataset

Initialization:

1: $i = 0$ (Index for i-th random walk)

Random walk iterates:

1: **repeat**
2: $i \leftarrow i + 1$ (One walk proceed)
3: node m receives the model parameters $\{\mathbf{O}^{\star}_{l,r,m}\}_{l=0}^{l=L}$ from a neighboring node
4: Activates learning without forgetting on m-th node
5: $l = -1$ (Index for l-th layer)
6: **repeat**
7: $l \leftarrow l + 1$
8: Compute $\mathbf{Y}_{l,m} = \mathbf{g}(\mathbf{W}_l \mathbf{Y}_{l-1,m})$
9: Solve $\mathbf{O}^{\star}_{l,u,m}$ using (8)
10: **until** $l = L$
11: **until** $i = i_{max}$

IV. Experimental Results

We provide simulation results to evaluate the performance of three datasets, Vowel, NORB, and MNIST. Note that we include Vowel as it is challenging for LwF-dPLN due to the limited number of training samples. We use the Q-dimensional target vector $\mathbf{t}$ in a classification task as a discrete variable with an indexed representation of 1-out-of-Q-classes. A target variable (vector) instance has only one scalar component that is 1, and the other scalar components are zero. For the local PLN, we set the number of layers $L = 10$, the number of hidden neurons $h = 2Q + 100$, and $\epsilon = (2Q)^{\frac{1}{2}}$ for each layer. We randomly distribute training data among processing nodes in the network and test the proposed learning algorithm in various conditions.

We use a circular communication network system for the experiments, we denote $\mathcal{N}_m$ as the set of neighbors with whom the m-th node is connected. Note that $m \notin \mathcal{N}_m$. In this setup, we assume that there is no master node and no node is isolated either. This implies the network policy matrix is chosen in such a way that every node is connected to its neighbors with communication resources. That means every neighbor of m-th node has the same possibility to receive parameters from m-th node, which is $p_{m \to n} = \frac{1}{|\mathcal{N}_m|}$, where $|\mathcal{N}_m|$ denotes cardinality of the set $\mathcal{N}_m$. In a circular communication network, every node has the same number of neighbors. Therefore we can denote d as the network degree, which has the follow relation with $p_{m \to n}$ and $|\mathcal{N}_m|$

$$p_{m \to n} = \frac{1}{|\mathcal{N}_m|} = \begin{cases} \frac{1}{2d}, & d < d_{max} \\ \frac{1}{M-1}, & d = d_{max} \end{cases}.$$

TABLE I: Classification performance comparison between dPLN with centralized equivalence, LwF-dPLN, and local PLN on a circular graph where $M = 10$, $d = 2$, $K = 100$

Dataset	dPLN		LwF-dPLN		local PLN
	Test Accuracy	Training Time(s)	Test Accuracy	Training Time(s)	Test Accuracy
Vowel	56.0±0.3	10.86	38.0±2.6	1.63	24.8±2.9
NORB	82.5±0.3	36.07	81.6±0.9	25.94	80.4±0.5
MNIST	91.5±0.2	38.07	89.7±0.2	14.45	87.7±0.3

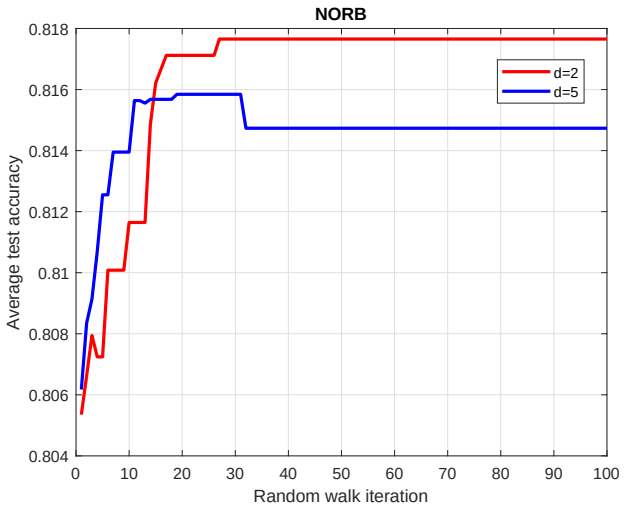

Fig. 1: Average test accuracy versus random walk iterations for $d = 2$, $d = 5$, NORB dataset, $M = 10$.

We test the algorithm with a different number of processing nodes, and different network degree d, i.e., each processing node has exactly d neighbors. The classification performance results are reported in Table I, the corresponding hyperparameters are found by cross-validation using local data. We compare the performance of LwF-dPLN with local PLN when no knowledge communication happens in the network. We also provide the upper bound of the performance for distributed PLN, which has the centralized equivalence, i.e. the case where all the data is available at one processing node.

Progressive performance. From the simulation results shown in Table I, we observe that the proposed algorithm provides a reasonable performance drop comparing with dPLN with centralized equivalence, and outperforms the local PLN without random communications. For efficiency concern, compared with dPLN, LwF-dPLN enjoys efficient computation by allowing real-time utilization of the local learning model and releasing nodes from waiting for up-to-date information from neighbors.

Efficient communication. We fix the number of nodes $M = 10$, and test the performance of LwF-dPLN on different network degree d for NORB dataset. The test accuracy versus random walk iteration is shown in Figure 1. The average test accuracy requires a very similar number of random walk iterations to converge, which implies that the proposed algorithm

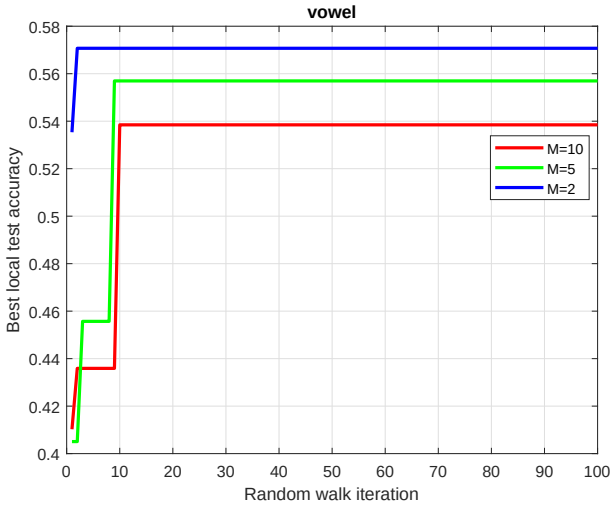

Fig. 2: Best local test accuracy versus random walk iterations for $M = 2, M = 5, M = 10$, Vowel dataset, ring network.

relaxes the condition of communication.

Behaviour for limited data. The Vowel dataset is challenging because it contains a limited amount of training data. The total number of training data points is 594. Therefore when M is 10, each node has around 60 data points. From Table I, we observe performance improvement for LwF-dPLN than the local PLN. Still there is a significant loss compared to the centralized dPLN. In this respect, we show an interesting observation - how local performance among the nodes improves over random walk iteration for information exchange in training LwF-dPLN. We test the learning performance for the Vowel dataset, on a ring network for different numbers of nodes. A ring network has $d = 1$. As shown in figure 2, the best local PLN converges to a satisfying level of 54%, which is close to the centralized performance. That means random walk helps to generate a few good models in some nodes.

V. Conclusion

We develop an incremental learning algorithm This work adapts learning without forgetting (LwF) to a decentralized setup by random walk based communication. We apply the algorithm on a feed forward neural network PLN, the experiments show that learning without forgetting decentralized PLN (LwF-dPLN) step-wisely improves the local learning model and converges to a competitive level as the centralized equivalent dPLN. The proposed algorithm requires very few communication resources and releases individual nodes from waiting for knowledge exchange from neighboring nodes.

References

[1] M. Kamp, L. Adilova, J. Sicking, F. Hüger, P. Schlicht, T. Wirtz, and S. Wrobel, "Efficient Decentralized Deep Learning by Dynamic Model Averaging," *arXiv preprints arXiv:1807.03210*, Jul 2018.

[2] N. Lewis, S. Plis, and V. Calhoun, "Cooperative learning: Decentralized data neural network," *International Joint Conference on Neural Networks (IJCNN)*, pp. 324–331, May 2017.

[3] A. Shirazinia, S. Dey, D. Ciuonzo, and P. Salvo Rossi, "Massive mimo for decentralized estimation of a correlated source," *IEEE Transactions on Signal Processing*, vol. 64, no. 10, pp. 2499–2512, May 2016.

[4] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 285–292, May 2017.

[5] S. Gupta, W. Zhang, and F. Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, Dec 2016, pp. 171–180.

[6] 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. [Online]. Available: http://arxiv.org/abs/1602.06709

[7] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Advances in Neural Information Processing Systems 30*.

[8] X. Li, G. Zhang, K. Li, and W. Zheng, "Chapter 4 - deep learning and its parallelization," in *Big Data*, R. Buyya, R. N. Calheiros, and A. V. Dastjerdi, Eds. Morgan Kaufmann, 2016, pp. 95 – 118. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128053942000040

[9] 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.

[10] 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*, Sep. 2017.

[11] X. Liang, A. M. Javid, M. Skoglund, and S. Chatterjee, "Distributed large neural network with centralized equivalence," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2976–2980, April 2018.

[12] ——, "Asynchronous decentralized learning of a neural network," *Accepted to IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2020.

[13] 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.

[14] 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.

[15] N. Bastianello, R. Carli, L. Schenato, and M. Todescato, "Asynchronous distributed optimization over lossy networks via relaxed ADMM: Stability and linear convergence," *arXiv preprint arXiv:1901.09252*, 2019.

[16] Z. Peng, Y. Xu, M. Yan, and W. Yin, "ARock: An algorithmic framework for asynchronous parallel coordinate updates," *SIAM Journal on Scientific Computing*, vol. 38, no. 5, pp. A2851–A2879, 2016.

[17] V. Losing, B. Hammer, and H. Wersing, "Incremental on-line learning: A review and comparison of state of the art algorithms," *Neurocomputing*, vol. 275, pp. 1261 – 1274, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231217315928

[18] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017. [Online]. Available: https://www.pnas.org/content/114/13/3521

[19] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, and Y. Fu, "Large scale incremental learning," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[20] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *Advances in Neural Information Processing Systems 30*.

[21] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 12, pp. 2935–2947, Dec 2018.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*.