Federated Multiple Dataset Learning Using Levenberg Marquardt Algorithm

Sedat Akbal
Dept. of Computer Engineering
Hacettepe University
Ankara, Turkey
sedatakbal@gmail.com

Mehmet Önder Efe
Dept. of Computer Engineering
Hacettepe University
Ankara, Turkey
onderefe@gmail.com

***Abstract*—Neural networks have emerged as a pivotal machine learning paradigm within contemporary artificial intelligence research, particularly for their adeptness at modeling complex input-output relationships. This paper addresses the training of shallow neural networks, focusing on optimizing the widely used Levenberg-Marquardt (LM) algorithm. While the LM algorithm facilitates faster convergence through adaptive learning rates and the use of Jacobian matrices, it also presents challenges in computational complexity and memory usage due to the growth of Jacobian matrices with network and dataset dimensions. To mitigate issues such as catastrophic forgetting and computational complexity and memory usage due to the growth of Jacobian matrices, this study proposes a Federated Multiple Dataset Levenberg-Marquardt (fmLM) algorithm. By employing distinct mask vectors for each dataset, the fmLM enables selective activation of network weights, allowing simultaneous training across multiple datasets while maintaining performance. Experimental results demonstrate that the fmLM algorithm achieves a promising error levels after training, effectively managing the balance between weight sharing and retention of prior learning. This approach not only enhances the efficiency of neural network training but also contributes a novel solution to catastrophic forgetting, underscoring the potential for improved generalization in neural networks.**

***Index Terms*—Neural Network, Levenberg Marquardt, Multiple Data Learning, Federated Learning.**

I. Introduction

The field of neural networks is a prominent machine learning paradigm that play a crucial role in contemporary artificial intelligence research and applications. These networks, particularly in deep architectures, stand out for their ability to model and generalize complex input-output relationships [1]–[3]. In the learning processes of neural networks, it is essential to adjust the network's parameters, weights and biases, using appropriate optimization techniques [1], [2], [4], [5]. In neural networks, gradient descent and error backpropagation are commonly used. These algorithms are first-order methods that typically lead to long learning processes. In addition to these, there are faster second-order algorithms. Levenberg-Marquardt (LM) algorithm, as a second-order method, is a widely used optimization technique for training neural networks. The LM algorithm combines the advantages of the Gauss-Newton method and gradient descent, offering a faster and more stable convergence process [4]–[8]. LM algorithm dynamically adjusts the learning rate according to the structure of the error surface, thereby enhancing the efficiency of the training process [1], [5], [8]–[10]. The LM algorithm uses a Jacobian matrix in its computations to achieve rapid convergence. During the computation process, this matrix must be stored, transposed, and inverted. The use of Jacobian matrix, which grows in size proportionally to the cardinality of the parameter space and the number of network outputs, enables discovering the best path towards the global minimum; however, this also introduces high computational complexity and significant memory requirements as a disadvantage [1], [6]–[9]. Due to the speed and stability of the LM algorithm, various studies have been conducted in the literature to enhance the algorithm's efficiency by reducing its disadvantages. These studies include optimization of matrix operations [7], reduction of the dimensions of Jacobian matrices [6], [11], parallelization of computations in LM using vector operations available in processors [9], and optimization of momentum calculations to avoid local minima [5].

In addition to optimization efforts in the training processes of neural networks, there are also various studies focused on the development of multifunctional neural networks [1], [2], [12]. Training multiple datasets on the same neural network can be considered important for enhancing the network's generalization capabilities, efficiency, and effective utilization of resources [1], [2], [6], [10], [12]. Sequentially processing of the available data leads to a problem known in the literature as catastrophic forgetting. Catastrophic forgetting causes a decline in the network's performance on previous tasks and a reduction in its generalization ability [1]–[3], [10]–[12]. As a solution to catastrophic forgetting, the literature includes studies focused on masking certain parts of the network and preventing their modification [1], [2], [10]–[12].

As observed in studies addressing catastrophic forgetting, it can be stated that certain parts of the neural network are sufficient to prevent the forgetting of previous learning, while a portion of the network can be utilized for new learning. This study proposes a novel method to enhance the efficiency of the LM algorithm by masking specific parts of the neural network for different datasets, based on methods identified to address catastrophic forgetting.

In the proposed new method, the optimization of training processes for multiple datasets and the prevention of catastrophic forgetting are achieved by utilizing federated learning techniques. Federated learning is an approach that allows multiple clients to collaboratively train a machine learning model without sharing their local data [13]–[16]. In federated learning, raw data is stored on local devices or within organizations, and only model updates are shared via a central server or coordinator [13], [14], [17]. Model training is conducted on distributed datasets, which may be located in different organizations, devices, or locations [13], [14], [16]–[18]. Furthermore, federated learning can be effectively utilized in environments where data is heterogeneous [14], [17], [18].

In the method developed in this study, the ability of federated learning to be used in training multiple different datasets has been combined with masking technique used for optimizing different section of a neural networks for multiple tasks. The study presented in [1] is the most closely related work to the current research. Similar to the present study, the work in [1] also utilizes a masking technique combined with the LM algorithm to facilitate the training of multiple datasets on a shared network architecture. In the study [1], the network was trained using a single Jacobian matrix that incorporated all data from the datasets. Therefore, the size of the Jacobian matrix increases proportionally with the number of datasets and the amount of data associated with each dataset. In contrast to study [1], the proposed novel method herein employs the dataset-specific masking mechanism not only to manage multiple datasets but also to reduce the dimensionality of the Jacobian matrix by by excluding the deactivated weights from its columns. In the developed method, the masked segments of the neural network are treated as distinct clients, analogous to those in federated learning, and are trained independently. Following training on separate datasets, the resulting weights are combined using the federated averaging technique commonly adopted in federated learning frameworks [13], [15], and the global network weights are updated. The detailed explanation of the weight update procedure is provided in the second section.

The organization of this paper is as follows. Section 2 explains the proposed federated multiple dataset LM algorithm, Section 3 illustrates the proposed multiple dataset LM algorithm through an example problem, and Section 4 presents the results of the study.

II. Federated Multiple Dataset Levenberg-Marquardt Algorithm (fmLM)

The fmLM developed within this study uses separate mask vectors for each dataset to be trained. The weights of the network for each dataset are either enabled or disabled with the created mask vectors (algorithm 1). The algorithm developed for generating the mask vectors includes parameters that allow for targeted control, enabling the allocation of varying proportions to different section of the network. Observations from the experiments conducted during training with the fmLM algorithm (algorithm 2) revealed that mask vectors, designed to progressively reduce the shared use of weights across datasets from the input layer to the output layer, have a positive impact on the training speed. An example network structure created with the developed algorithm is shown in the Fig. 1.

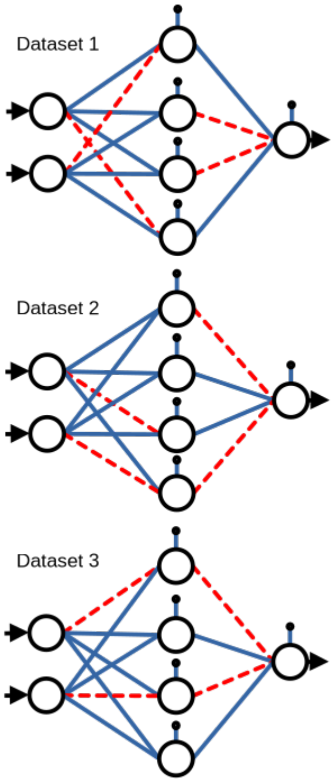

Fig. 1. Three datasets and preselected three subnetworks. Inactive parameters are shown dashed and the weight values are frozen throughout the NN

The figure is generated for three datasets. Solid lines denote weights that are active during model computation for related datasets, while dashed lines correspond to weights that are inactive from the learning process for related datasets in this context. As can be seen from the figure, the number of weights commonly used from the input to the output decreases. The mask creation algorithm ensures that each parameter is active for at least one dataset and guarantees that every input is connected to the output.

When training the neural network, after the mask vectors ($\mathbf{m_d}$) are created, each dataset's mask vector is multiplied with the weight vector in sequence. The weights corresponding to zero in the mask vector are deactivated for training. A forward propagation is then performed on the network using the masked weight values, and the error values (e_d^i)(1) for each output of each dataset ($\mathbf{d}$) were calculated by subtracting the neural network's outputs (y_d^i) from the target outputs (τ_d^i). Error vector ($\mathbf{e_d}$)(2) is generated for each dataset by using error values (1).

$$e_d^i = \tau_d^i - y_d^i \tag{1}$$

$$\mathbf{e_d} = \begin{bmatrix} e_d^1 \\ \vdots \\ e_d^i \end{bmatrix} \tag{2}$$

Algorithm 1 Creating Network Weight Masking Algorithm

```
 1: Get mask percentage value ($percent$)
 2: Get max value
 3: Get min value
 4: {The max and min values represent the maximum and
    minimum number of datasets that are allowed to share
    weights within the network. }
 5: if  a layer has only 1 weight then
 6:     Set mask value of one to all data set
 7: else
 8:     Generate a permutation vector($per[]$) with the same
        dimensionality as the parameter vector
 9:     for a in permutation_vector do
10:         if $per[a] < percent$ then
11:             Randomly select a dataset and assign the corre-
                sponding mask value at index $a$ to 1.
12:         else
13:             Select a random number within the range of the
                minimum and maximum values.
14:             Randomly choose that many datasets, and assign
                the corresponding mask value at index a to 1 for
                each selected dataset.
15:         end if
16:     end for
17: end if
```

By using the generated error vector (1) and the masked weight vector (3) for parameters ($\mathbf{p}$), the Jacobian matrix (J_d) (4) is formed for each dataset ($\mathbf{d}$) separately. Mask vector contains 0 and 1 values. When 0 values are multiplied by the corresponding parameter, that corresponding parameter is deactivated for the corresponding dataset $\mathbf{d}$.

$$\mathbf{m_d} = \begin{bmatrix} m_d^1, & m_d^2, & m_d^3, \cdots & m_d^p \end{bmatrix} \tag{3}$$

$$J_d = \begin{bmatrix} m_d^1 \cdot \frac{\partial e_d^1}{\partial w_1} & m_d^2 \cdot \frac{\partial e_d^1}{\partial w_2} & \cdots & m_d^p \cdot \frac{\partial e_d^1}{\partial w_p} \\ m_d^1 \cdot \frac{\partial e_d^2}{\partial w_1} & m_d^2 \cdot \frac{\partial e_d^2}{\partial w_2} & \cdots & m_d^p \cdot \frac{\partial e_d^2}{\partial w_p} \\ \vdots & \vdots & \ddots & \vdots \\ m_d^1 \cdot \frac{\partial e_d^i}{\partial w_1} & m_d^2 \cdot \frac{\partial e_d^i}{\partial w_2} & \cdots & m_d^p \cdot \frac{\partial e_d^i}{\partial w_p} \end{bmatrix} \tag{4}$$

To reduce the size of the Jacobian matrix, columns corresponding to the deactivated weights are discarded and they are not included as in (5), so the reduced Jacobian (J_{d_r}) which calculated for each corresponding dataset $\mathbf{d}$ has fewer columns.

$$J_{d_r} = \begin{bmatrix} \frac{\partial e_d^1}{\partial w_1} \frac{\partial e_d^1}{\partial w_3} & \cdots & \frac{\partial e_d^1}{\partial w_{10}} \frac{\partial e_d^1}{\partial w_{13}^i} & \cdots & \frac{\partial e_d^1}{\partial w_p} \\ \frac{\partial e_d^2}{\partial w_1} \frac{\partial e_d^2}{\partial w_3} & \cdots & \frac{\partial e_d^2}{\partial w_{10}} \frac{\partial e_d^2}{\partial w_{13}^i} & \cdots & \frac{\partial e_d^2}{\partial w_p} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial e_d^i}{\partial w_1} \frac{\partial e_d^i}{\partial w_3} & \cdots & \frac{\partial e_d^i}{\partial w_{10}} \frac{\partial e_d^i}{\partial w_{13}} & \cdots & \frac{\partial e_d^i}{\partial w_p} \end{bmatrix} \tag{5}$$

Algorithm 2 Federated Multiple Dataset LM Algorithm

```
 1: Get input vector
 2: Get target vector
 3: Create mask vector for each dataset
 4: Set weight and bias vector randomly
 5: for 1 to epoch do
 6:     for 1 to number_of_dataset do
 7:         Mask the weight vector with mask vector
 8:         for 1 to input_size do
 9:             Perform forward propagation with the masked
                weights
10:         end for
11:         Calculate error value
12:         Calculate MSE (Mean Squared Error)
13:         Calculate Jacobian matrix with masked weights
14:     end for
15:     while True do
16:         for 1 to number_of_data do
17:             Perform forward propagation with the masked
                weights
18:         end for
19:         Remove the zero value colunms from jacobian
20:         Calculate $\Delta\mathbf{w} = \left(\mathbf{J_{d_r^i}}^\mathsf{T}\mathbf{J_{d_r^i}} + \mu\mathbf{I}\right)^{-1}\mathbf{J_{d_r^i}}^\mathsf{T}\mathbf{e}$
21:         {Calculate the temporary w values to test the perfor-
            mance.}
22:         calculate $\mathbf{w_{t+1}} = \mathbf{w_t} - \Delta\mathbf{w}$
23:         for 1 to number_of_data do
24:             {to check new weights performance}
25:             Perform forward propagation with the masked
                weights
26:         end for
27:         Calculate error value
28:         Calculate MSE
29:         Add new delta value to federated delta vector
30:         for 1 to number_of_data do
31:             Perform forward propagation with the masked
                weights
32:         end for
33:         Calculate error value
34:         Calculate MSE
35:         for d=1 to number_of_dataset do
36:             if $\mathbf{e_d}$ >= previous $\mathbf{e_d}$ then
37:                 {Compare error for each data set with their
                    previous error}
38:                 Increase lambda value
39:                 {Only increase the errors of the worse data }
40:             else
41:                 break While loop
42:             end if
43:         end for
44:     end while
45:     Decrease lambda value of all data set
46:     Display the min error
47:     Update weight parameter with $\Delta\mathbf{w_f}$
48: end for
```

With the reduced Jacobian, delta values ($\Delta\mathbf{w}_d$) in (6) are computed, and only the active weights are updated for the corresponding dataset for testing the new weights (7).

$$\Delta\mathbf{w}_d = \left(\mathbf{J_{d_r}}^\intercal\mathbf{J_{d_r}} + \mu_d\mathbf{I}\right)^{-1}\mathbf{J_{d_r}}^\intercal\mathbf{e}_d \tag{6}$$

$$\mathbf{w}_d(test) = \mathbf{w}_d(t) - \Delta\mathbf{w}_d \tag{7}$$

After the updates for all datasets, a second forward propagation is performed to check whether the error value of the new weights has decreased. This process is repeated sequentially for all datasets. If the error value for a dataset is equal to or greater than its previous value, the μ_d values for those datasets are increased; otherwise, they remain unchanged. This process is repeated until the error values for all datasets is smaller than their previous values. Once the error values of all datasets are smaller than their previous values, the μ_d value for each dataset is reduced. The delta values for the weights are then summed up to create the federated weight delta vector (ΔW_f) (8).

$$\Delta W_f = \sum_{d=1}^{n}\Delta\mathbf{w}_d \tag{8}$$

When constructing the federated delta weight vector, each dataset adds only the active weights corresponding to itself. Additionally, the number of datasets that contribute to each weight is stored in the federated mask vector (M_f) (9).

$$M_f = \sum_{d=1}^{n}\mathbf{m_d} \tag{9}$$

Each weight in the federated delta weight vector (10) is divided by the corresponding value in the federated mask vector, and the weights of the entire network are updated accordingly (11).

$$\Delta W_f \leftarrow \frac{\Delta W_f}{M_f} \tag{10}$$

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \Delta W_f \tag{11}$$

The training of the network continues until the specified performance value is reached or until a predefined stopping condition is met. The method applied here is based on federated averaging [19] and distributed training strategies [3]. This approach allows the proposed algorithm to train the network with datasets of varying sizes. Additionally, it prevents the size of the Jacobian matrix from growing proportionally as the number of datasets increases. As in federated learning, averaging the weights of the datasets in each iteration also provides a solution to the catastrophic forgetting problem.

III. Simulation Study

In the studied application, to test the developed fmLM algorithm, three function, sinc function (12), the Gaussian function (13), and the cosine function (14), were chosen and made orthogonal to each other using the Gram-Schmidt orthogonalization technique.

$$f_1(x_1,x_2) = \text{sinc}(x_1)\text{sinc}(x_2) \tag{12}$$

$$f_2(x_1,x_2) = e^{-(x_1^2+x_2^2)} \tag{13}$$

$$f_3(x_1,x_2) = \cos(\pi x_1)\cos(\pi x_2) \tag{14}$$

The function set is orthogonal in the range $(x_1,x_2) \in [-1,+1]\times[-1,+1]$. The goal of using orthogonal functions is to test the performance of the developed fmLM algorithm on data sets that may show considerable differences in a realistic scenario. For testing, a neural network with a 2-300-1 structure was used. The input vector has two input values $[x_1,x_2]$. Hidden layer has neurons having hyperbolic tangent activation function, while the output layer has a linear activation function. For each function, a total of 900 data sets were generated, with 30 data points per axis linearly. The same dataset was used as input for each function. Each function produces different outputs with the same dataset. To test the performance of the developed fmLM algorithm, a very low error rate ($9.0e-6$) was set as the performance target, and the error value reached by the algorithm after 2000 iterations was observed. The minimum mean squared error (MSE) value used for the error calculation is shown in equation 15 (M is the number of input in each dataset).

$$MSE = \frac{1}{M}e^Te \tag{15}$$

It was observed that the developed fmLM algorithm reached an MSE value of $9.0e-6$ before completing 2000 iterations. However, repeated trials revealed that the number of iterations required to reach an MSE value of $9.0e-6$ exhibited significant variability, ranging from 40 to 1900 iterations. This variability can be attributed to the random selection of weights when creating the mask vectors. Due to the assignment of different random weights in each new training cycle, the distinct characteristics of the datasets may sometimes result in conflicts.

After training the developed fmLM algorithm, for comparison, the outputs of the target functions are represented in Fig. 2 and the outputs of fmLM algorithm with the error surface between the target values and the model results are represented in Fig. 3, Fig. 4, and Fig. 5.

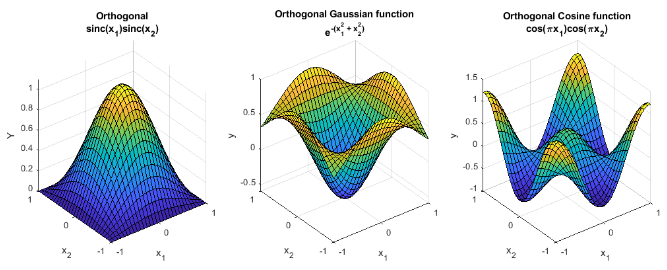

Fig. 2. Target datasets

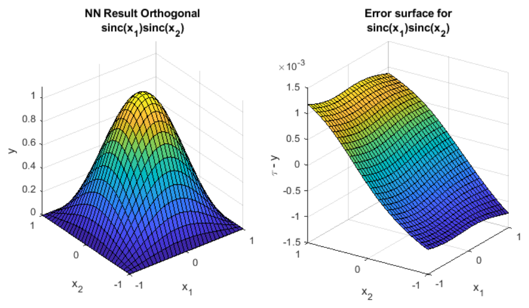

Fig. 3. Left: model results for sinc function. Right: The error surface between target and model result τ-y

Additionally, to investigate the impact of federated training on performance, the same datasets were compared with the non-federated LM algorithm, where all datasets are trained using a single Jacobian matrix.

Fig. 6 and Fig. 7 illustrate the comparative performance of the fmLM and non-federated LM algorithms during training with varying input sizes (900, 1600, and 2500). Fig. 8 and Fig. 9 present a comparative analysis of the performance of the fmLM and non-federated LM algorithms during training with different numbers of neurons (300, 600, and 900). As can be inferred from Fig. 6 and Fig. 8, the fmLM algorithm utilizes less memory compared to the non-federated LM algorithm. Specifically, as shown in Figure 8, the network's performance gain increases with the number of neurons, reaching 50% when the neuron count reaches 900. Based on Fig. 6 and Fig. 8, it can be stated that the number of neurons — and consequently the number of weights associated with them — has a greater impact on memory gain than the number of inputs.

As depicted in Fig. 7 and Fig. 9, the fmLM algorithm not only demonstrates improved memory efficiency but also accelerates neural network training by substantially reducing the required number of epochs. This can be attributed to the fact that, in the fmLM algorithm, each dataset independently attempts to guide the network's weights toward its own minimum loss, and distinct lambda (μ_d) values are used for each dataset during separate training phases. As a result, it may navigate the loss surface more effectively and reach a lower overall minimum.

Fig. 10 shows the number of epochs required by the fmLM and non-federated LM algorithms to reach the same error level over 10 different training runs. For each training run, both networks used the same inputs, initial weight values, and mask vectors; however, in each new run, the mask vectors were randomly reinitialized. As can be inferred from the Fig. 10, weight masking has a significant impact on the performance of the learning processes. Instead of applying masking randomly, employing methods that dynamically mask weights based on the characteristics of the datasets is expected to have a substantial positive effect on performance.

Upon evaluating the observed results, it can be concluded that the developed fmLM algorithm is capable of learning from multiple datasets simultaneously and is not affected by the

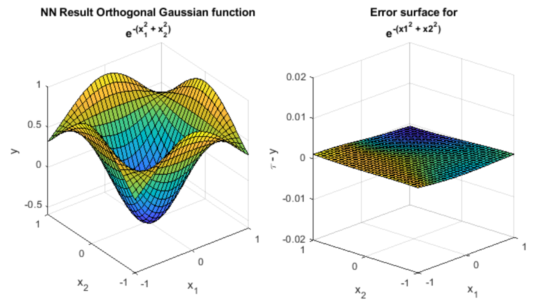

Fig. 4. Left: model results for Gaussian function. Right: The error surface between target and model result τ-y

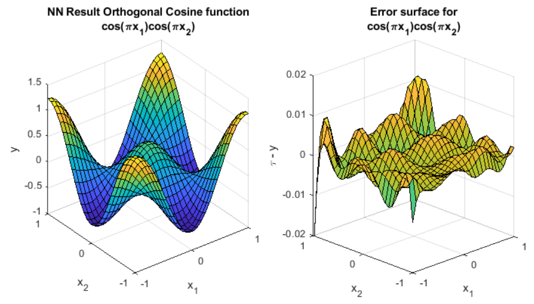

Fig. 5. Left: model results for cosine function. Right: The error surface between target and model result τ-y

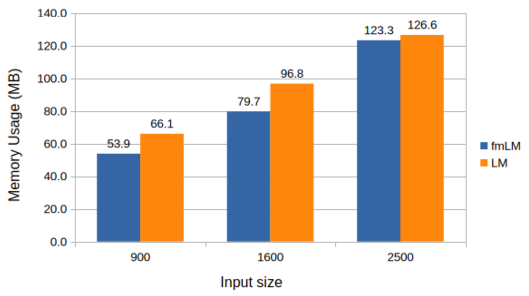

Fig. 6. fmLM (federated LM), LM (non-federated LM)

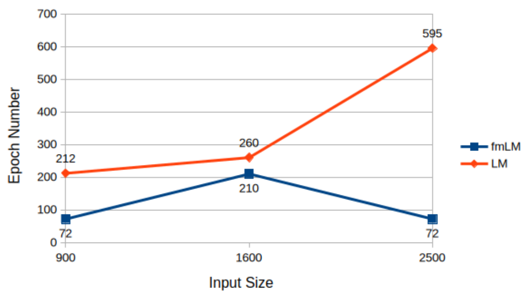

Fig. 7. fmLM (federated LM), LM (non-federated LM)

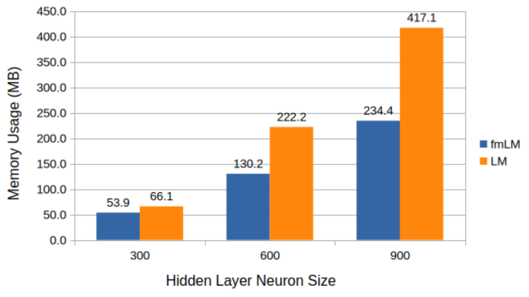

Fig. 8. fmLM (federated LM), LM (non-federated LM)

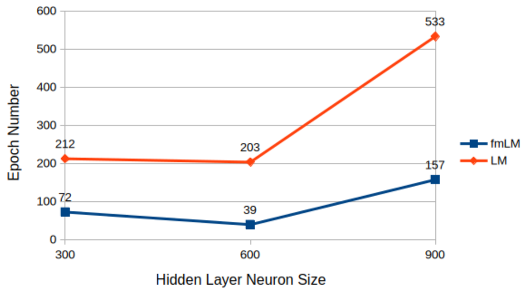

Fig. 9. fmLM (federated LM), LM (non-federated LM)

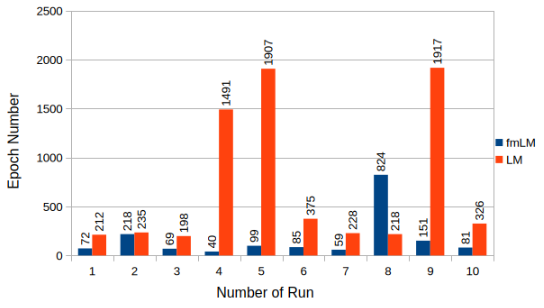

Fig. 10. fmLM (federated LM), LM (non-federated LM)

phenomenon of catastrophic forgetting.

IV. Conclusion

This study proposes a modification to the Levenberg-Marquardt algorithm for training a neural network with multiple datasets. The proposed approach enables the simultaneous training of multiple datasets and prevents the issue of catastrophic forgetting. Additionally, the approach allows the training of datasets with varying sizes with a little modification. When creating masks for different datasets, the number of weights shared across datasets is gradually reduced from input to output layers. This gradual reduction in shared weights has been observed to positively impact the performance of neural network training, indicating that the approach introduced for mask creation offers a novel solution.

References

[1] M. Ö. Efe, B. Kürkçü, C. Kasnakoğlu, Z. Mohamed, and Z. Liu, "A Modified Levenberg Marquardt Algorithm for Simultaneous Learning of Multiple Datasets," *IEEE Transactions on Circuits and Systems II-Express Briefs*, vol. 71, no. 8, pp. 2379-2383, April 2024.

[2] M. Ö. Efe, B. Kürkçü, C. Kasnakoğlu, Z. Mohamed, and Z. Liu, "Switched Neural Networks for Simultaneous Learning of Multiple Functions," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 8, no. 6, pp. 3095-3104, 2024.

[3] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Aguera y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," *International Conference on Artificial Intelligence and Statistics*, pp. 1273–1282, Apr. 2017, [Online]. Available: http://proceedings.mlr.press/v54/mcmahan17a/mcmahan17a.pdf

[4] M. T. Hagan and M. B. Menhaj, "Training Feedforward Networks with the Marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989-993, Nov. 1994.

[5] J. S. Smith, B. Wu, and B. M. Wilamowski, "Neural Network Training with Levenberg–Marquardt and Adaptable Weight Compression," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 2, pp. 580–587, Feb. 2019

[6] S. B. Shinde and S. S. Sayyad, "Cost Sensitive Improved Levenberg Marquardt Algorithm for Imbalanced Data," *IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)* vol. 6, pp. 1–4, Dec. 2016.

[7] B. M. Wilamowski and H. Yu, "Improved Computation for Levenberg-Marquardt Training," *IEEE Transactions on Neural Networks*, vol. 21, no. 6, pp. 930–937, Jun 2010.

[8] G. Zhou and J. Si, "Advanced Neural-Network Training Algorithm with Reduced Complexity Based on Jacobian Deficiency," *IEEE Transactions on Neural Networks*, vol. 9, no. 3, pp. 448–453, May 1998.

[9] Bilski, Jacek Smolag, B. Kowalczyk, Konrad Grzanek, and I. Izonin, "Fast Computational Approach to the Levenberg-Marquardt Algorithm for Training Feedforward Neural Networks," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 13, no. 2, pp. 45–61, Mar. 2023.

[10] K. Shmelkov, C. Schmid, and K. Alahari, "Incremental Learning of Object Detectors without Catastrophic Forgetting," *IEEE International Conference on Computer Vision (ICCV)*, pp. 3420–3429, Oct. 2017.

[11] J. Bilski, B. Kowalczyk, and K. Grzanek, "The Parallel Modification to the Levenberg-Marquardt Algorithm," *Lecture Notes in Computer Science*, pp. 15–24, 2018. doi:10.1007/978-3-319-91253-0_2.

[12] A. Mallya and S. Lazebnik, "PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, pp. 7765-7773, Jun. 2018.

[13] Q. Li, Z. Wen, Z.M. Wu, S.X. Hu, N.B. Wang, Y. Li, X. Liu, and B.S. He, "A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 1–1, 2023.

[14] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, et al., "Advances and Open Problems in Federated Learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1, 2021.

[15] Y. Hoshino, H. Kawakami, and H. Matsutani, "Federated Learning of Neural Ode Models with Different Iteration Counts," *IEICE Transactions on Information and Systems*, vol. E107.D, no. 6, pp. 781–791, Jun. 2024.

[16] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated Machine Learning: Concept and Application," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1-19, Feb. 2019.

[17] J. Konečný, H. B. Mcmahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated Learning: Strategies for Improving Communication Efficiency," arXiv:1610.05492v2, (Cornell University), Oct. 2017.

[18] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated Optimization: Distributed Machine Learning for On-Device Intelligence," arXiv preprint arXiv:1610.02527, 2016. [Online]. Available: https://arxiv.org/abs/1610.02527.

[19] 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, pp. 456-464, 2010.