Adaptive Learning without Forgetting via Low-Complexity Convex Networks

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

***Abstract*—We study the problem of learning without forgetting (LwF) in which a deep learning model learns new tasks without a significant drop in the classification performance on the previously learned tasks. We propose an LwF algorithm for multilayer feedforward neural networks in which we can adapt the number of layers of the network from the old task to the new task. To this end, we limit ourselves to convex loss functions in order to train the network in a layer-wise manner. Layer-wise convex optimization leads to low-computational complexity and provides a more interpretable understanding of the network. We compare the effectiveness of the proposed adaptive LwF algorithm with the standard LwF over image classification datasets.**

***Index Terms*—learning without forgetting, convex neural networks, size adaptive, low complexity**

I. Introduction

The machine learning literature has been enriched by neural networks in the past decade, pushing the state-of-the-art performance and outperforming the traditional methods in different areas, e.g., image denoising, object classification, etc [1]–[3]. A trained neural network can achieve good classification performance if it is tested on the same object classes which was trained for. However, if the number of classes (tasks) increases, the network must be updated in a way that can preserve the knowledge learned from old classes as well. This problem is referred to as incremental learning (IL) in the literature [4].

In recent years, IL has attracted growing attention in the machine learning community, finding various applications in big data processing [5], robotics [6], automated annotation [7], etc. In this setup, training the network by only using the new task data, may lead to so-called *catastrophic forgetting*, in which the network most likely forgets the old task if the new task is sufficiently dissimilar [8], [9]. Therefore, there is a need to incorporate, at least, some information from old task data in the training process. In this light, the works in the literature can be divided into two different categories: methods that use samples of the old task, and methods that do not use samples of the old task in the learning process.

In the first category, the common approach is to use a small percentage of the old dataset $\mathcal{D}_o$ while using the new dataset $\mathcal{D}_n$ to train the network [10]–[12]. However, incrementally storing the old tasks data increases memory requirements and therefore, is not practical when the memory resources are limited. In some applications such as distributed learning, sharing the old data even raises serious privacy and security concerns [13]. The second category of algorithms solves these issues [8], [14]–[16]. A prominent approach in this class of algorithms is *learning without forgetting* (LwF) [14] and its variants, such as LwF-MC [11]. The main idea in LwF is to keep the output of the old model for the new dataset "relatively" unchanged in the training process. In this way, one hopes to preserve some useful information about the old task in the network while achieving an acceptable performance on the new task. However, all of the above works require high computational resources to train deep neural network architectures such as CNN. Therefore, they might not be suitable for applications where memory and computational resources are limited, e.g., deep learning on the edge [17]. To the best of our knowledge, the only work that addresses low-complexity design in this setup is [12] which uses samples of the old dataset $\mathcal{D}_o$ to avoid catastrophic forgetting. We propose a low-complexity LwF scheme for a multilayer feedforward neural network that does not use the old task data.

Another shortcoming of the above works is the fact that they are limited to a fixed-size architecture when training for the old and new tasks. For example, consider a scenario where the old dataset $\mathcal{D}_o$ can be trained on a network with 5 layers without overfitting but after adding the new dataset $\mathcal{D}_n$, the network requires 10 layers to achieve a good performance. Limiting the architecture to have a fixed size during the training process may lead to lower performance in the old or new task. We propose a size-adaptive approach for LwF over a multilayer feedforward network by using layer-wise convex optimization.

Our contribution: We extend the LwF idea [14] to be applicable to size-adaptive multilayer feedforward neural networks. We refer to our proposed method as adaptive LwF throughout the manuscript. We implement our method on a recently proposed self size-estimating feedforward network (SSFN) [18]. This architecture employs a combination of the ReLU activation function and random weights to guarantee a monotonically decreasing training cost as the number of layers increases. This property is the key to develop an adaptive LwF scheme. We show that adaptive LwF enjoys low-computational complexity due to using a series of layer-wise convex optimizations. We test the effectiveness of the proposed method on several well-known classification datasets with different similarity levels between the old and new tasks. Finally, we show that a layer-wise LwF scheme leads to less sensitivity to the hyperparameters as the network gets deeper.

II. Preliminaries

A. Learning without Forgetting

We briefly overview the concept of learning without forgetting [14]. Assume the old dataset $(\mathbf{x}, \mathbf{t}) \in \mathcal{D}_o$, $\mathbf{x} \in \mathbb{R}^{P_o}$ and $\mathbf{t} \in \mathbb{R}^{Q_o}$, is used to train a neural network $f_{\mathbf{W}}(\cdot)$, where $\mathbf{W}$ represents the parameters of the network. Let us define the old trained weights as

$$\mathbf{W}_o^\star \in \arg\min_{\mathbf{W}} \mathcal{L}_o(\mathbf{t}, f_{\mathbf{W}}(\mathbf{x})) + \mathcal{R}(\mathbf{W}), \tag{1}$$

where $\mathcal{L}_o$ is the loss function over all samples of the old dataset $\mathcal{D}_o$, and $\mathcal{R}$ is the regularization term to avoid overfitting, e.g., a simple ℓ_2-norm weight decay.

We wish to update the trained network such that it be able to learn the mapping of the new dataset $(\mathbf{x}, \mathbf{t}) \in \mathcal{D}_n$, $\mathbf{x} \in \mathbb{R}^{P_n}$ and $\mathbf{t} \in \mathbb{R}^{Q_n}$, while maintaining a reasonable performance on the old dataset $\mathcal{D}_o$. To this end, the main concept in LwF idea is to keep the output of the old model for the new data, $\hat{\mathbf{t}}_o = f_{\mathbf{W}_o^\star}(\mathbf{x}),\ \mathbf{x} \in \mathcal{D}_n$, 'relatively' unchanged in the training process. Specifically, the update is done as follows

$$\mathbf{W}_n^\star, \mathbf{W}_o^\star \in \arg\min_{\mathbf{W}_n, \mathbf{W}_o} \mathcal{L}_n(\mathbf{t}, f_{\mathbf{W}_n}(\mathbf{x})) + \lambda \mathcal{L}_n(\hat{\mathbf{t}}_o, f_{\mathbf{W}_o}(\mathbf{x})) + \mathcal{R}(\mathbf{W}_n, \mathbf{W}_o), \tag{2}$$

where $\mathcal{L}_n$ is the loss function over all samples of the new dataset $\mathcal{D}_n$, $\mathbf{W}_n$ is the added task-specific parameters of the network for the new dataset $\mathcal{D}_n$, and λ is the loss balance weight which we refer to it as the 'forgetting factor' in the rest of manuscript. A larger value of λ favors the old task over the new task in the training process.

B. SSFN

We briefly discuss SSFN in this section for completeness. Details can be found in [18]. SSFN is a feedforward neural network that uses a combination of random weights and layer-wise optimization approach to deliver a low-complexity architecture. While the work of [18] developed the SSFN architecture that learns its parameters and size of the network automatically, we work with a fixed size SSFN and learn its parameters. The number of layers L and the hidden neurons for the l-th layer n_l are fixed a-priori. For simplicity, we assume that all layers have the same number of hidden neurons, which means $n_l = n, \forall l$.

The SSFN parameters $\{\mathbf{W}_l\}_{l=1}^L$ are learned layer-by-layer in a sequential forward learning approach. The feature vector of l-th layer is constructed as follows

$$\mathbf{y}_l = \mathbf{g}(\mathbf{W}_l\, \mathbf{g}(\ldots \mathbf{g}(\mathbf{W}_2\, \mathbf{g}(\mathbf{W}_1\, \mathbf{x}))\ldots)) \in \mathbb{R}^n. \tag{3}$$

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

1) For all the samples $(\mathbf{x}, \mathbf{t})$ in the training dataset $\mathcal{D}$, we compute $\mathbf{y}_l = \mathbf{g}(\mathbf{W}_l\, \mathbf{g}(\ldots \mathbf{g}(\mathbf{W}_1\, \mathbf{x})\ldots))$.
2) Using the feature vectors $\{\mathbf{y}_l\}$, we define a training cost $\mathcal{C}_l = \sum_{(\mathbf{x},\mathbf{t})\in\mathcal{D}} \|\mathbf{t} - \mathbf{O}\,\mathbf{y}_l\|_2^2$. We compute the optimal output matrix $\mathbf{O}_l$ by solving the convex optimization problem

$$\mathbf{O}_l^\star = \arg\min_{\mathbf{O}} \mathcal{C}_l \text{ s.t. } \|\mathbf{O}\|_F^2 \le \epsilon_l. \tag{4}$$

   It is shown in [18] that we can choose the regularization parameters $\epsilon_l = \epsilon = 2Q$, $l = 0, 1, 2, \ldots, L$. Note that $\mathbf{O}_0$ is a $Q \times P$-dimensional matrix, and every $\mathbf{O}_l$ for $l = 1, 2, \ldots, L$ is a $Q \times n$-dimensional amtrix.
3) We form the weight matrix for the $(l+1)$-th layer

$$\mathbf{W}_{l+1} = \begin{bmatrix} \mathbf{V}_Q \mathbf{O}_l^\star \\ \mathbf{R}_{l+1} \end{bmatrix} \in \mathbb{R}^{n\times n}, \tag{5}$$

   where $\mathbf{V}_Q = [\mathbf{I}_Q\ -\mathbf{I}_Q]^\top \in \mathbb{R}^{2Q\times Q}$ is a fixed matrix and $\mathbf{R}_{l+1}$ is an instance of random Gaussian matrix. Note that we only learn $\mathbf{O}_l^\star$ to form $\mathbf{W}_{l+1}$. After constructing the weight matrix according to (5), the $(l+1)$-layer network is $\mathbf{y}_{l+1} = \mathbf{g}(\mathbf{W}_{l+1}\mathbf{y}_l)$.

It is shown that the three steps mentioned above guarantee monotonically decreasing cost $\mathcal{C}_l$ with increasing layer number l, i.e., $\mathcal{C}_l \le \mathcal{C}_{l-1}$. The monotonically decreasing cost is the key to address the LwF optimization problem (2) in an adaptive layer-wise manner. Thus, before we proceed to design an adaptive learning without forgetting scheme, we present Proposition 2 of [18] in the following Remark.

Remark 1. *For a given $\epsilon_l \ge 2Q$ in the optimization* (4)*, the feasible point $\bar{\mathbf{O}} = [\mathbf{U}_Q\ \mathbf{0}]$ leads to $\mathcal{C}_l = \mathcal{C}_{l-1}$, where $\mathbf{0}$ is a zero matrix of size $Q \times n$ and $\mathbf{U}_Q = [\mathbf{I}_Q\ \ -\mathbf{I}_Q]$. Therefore, any other optimal point in the feasible set leads to $\mathcal{C}_l \le \mathcal{C}_{l-1}$.*

Proof. It is shown in Proposition 1 of [18] that for a single layer feedforward network with ReLU activation function $\mathbf{g}(\cdot)$, we have $\mathbf{U}_Q\mathbf{g}(\mathbf{V}_Q\mathbf{y}) = \mathbf{y}, \forall \mathbf{y} \in \mathbb{R}^Q$. In addition, by construction in equation (3), we know that $\mathbf{y}_l = \mathbf{g}(\begin{bmatrix} \mathbf{V}_Q\mathbf{O}_{l-1}^\star \\ \mathbf{R}_l \end{bmatrix} \mathbf{y}_{l-1})$. Therefore, we can conclude

$$\begin{aligned}
\mathcal{C}_l &= \sum_{(\mathbf{x},\mathbf{t})\in\mathcal{D}} \|\mathbf{t} - \bar{\mathbf{O}}\,\mathbf{y}_l\|_2^2, \\
&= \sum_{(\mathbf{x},\mathbf{t})\in\mathcal{D}} \|\mathbf{t} - [\mathbf{U}_Q\ \mathbf{0}]\,\mathbf{y}_l\|_2^2, \\
&= \sum_{(\mathbf{x},\mathbf{t})\in\mathcal{D}} \|\mathbf{t} - \mathbf{U}_Q\mathbf{g}(\mathbf{V}_Q\mathbf{O}_{l-1}^\star\mathbf{y}_{l-1})\|_2^2, \\
&= \sum_{(\mathbf{x},\mathbf{t})\in\mathcal{D}} \|\mathbf{t} - \mathbf{O}_{l-1}^\star\mathbf{y}_{l-1}\|_2^2 = \mathcal{C}_{l-1}.
\end{aligned} \tag{6}$$

Note that any other optimal point in the feasible set leads to a lower training cost compared to the previous layer because of the convexity of the optimization problem (4). □

III. Adaptive Learning without Forgetting

In this section, we propose a layer-wise update scheme for implementing the LwF idea in SSFN architecture. Consider the setup in Subsection II-A. We apply the update rule in equation (2) in every layer of SSFN in a layer-wise manner. In specific, we modify the optimization problem (4) to maintain the old task performance as the following.

Let us define the old task dataset $\mathcal{D}_o$ in the matrix form as $\mathbf{X}_o \in \mathbb{R}^{P_o \times N_o}$ and $\mathbf{T}_o \in \mathbb{R}^{Q_o \times N_o}$, where P_o and Q_o are the dimensions of the input and target, respectively, and N_o is the number of samples in the new dataset. Assume an SSFN with L_o layers is trained using the old dataset $\mathcal{D}_o$. In every layer of SSFN, the convex optimization problem (4) can be written in the regularized form as

$$\mathbf{O}_{lo}^{\star} = \underset{\mathbf{O}_o}{\arg\min} \; \|\mathbf{T}_o - \mathbf{O}_o \mathbf{Y}_{lo}\|_F^2 + \eta \|\mathbf{O}_o\|_F^2, \tag{7}$$

where $\mathbf{Y}_{lo}$ denotes the matrix of old feature vectors $\mathbf{y}_l$ in equation (3). Note that the regularization hyperparameter η requires cross-validation. After training of the network, the set of optimized weights of all the layers $\{\mathbf{O}_{lo}^{\star}\}_{l=0}^{L_o}$ is stored in the memory.

After receiving the new dataset $\mathcal{D}_n$, we wish to adapt the network to have L_n layers while preserving the old task performance as much as possible. Note that we assume $L_n \geq L_o$. The case of $L_n < L_o$ can be easily handled by truncating the old trained network, and therefore, it is not of particular interest in practice. Let us define $\mathbf{X}_n$ and $\mathbf{T}_n$ as the new data and target matrix, respectively. In layer $l \leq L_o$, we have access to $\{\mathbf{O}_{lo}^{\star}\}_{l=0}^{L_o}$, so the update rule in equation (2) can be simplified to the following convex form

$$\mathbf{O}_{ln}^{\star}, \mathbf{O}_{lo}^{\star} = \underset{\mathbf{O}_n, \mathbf{O}_o}{\arg\min} \; \|\mathbf{T}_n - \mathbf{O}_n \mathbf{Y}_{ln}\|_F^2 + \lambda \|\mathbf{O}_{lo}^{\star} \mathbf{Y}_{ln} - \mathbf{O}_o \mathbf{Y}_{ln}\|_F^2 + \mathcal{R}(\mathbf{O}_n, \mathbf{O}_o), \tag{8}$$

where $\mathbf{Y}_{ln}$ denotes the matrix of new feature vectors $\mathbf{y}_l$ in equation (3). After updating the first L_o layers of the network sequentially by using (8), we don't have access to the optimized weight of following layers $l > L_o$. Therefore, it is not possible to use (8) in order to preserve the old task performance on the subsequent layers. Motivated by the cost-preserving property of $\bar{\mathbf{O}} = [\mathbf{U}_Q \; \mathbf{0}]$ in Remark 1, we update the subsequent layers as follows

$$\mathbf{O}_{ln}^{\star}, \mathbf{O}_{lo}^{\star} = \underset{\mathbf{O}_n, \mathbf{O}_o}{\arg\min} \; \|\mathbf{T}_n - \mathbf{O}_n \mathbf{Y}_{ln}\|_F^2 + \lambda \|\bar{\mathbf{O}} \mathbf{Y}_{ln} - \mathbf{O}_o \mathbf{Y}_{ln}\|_F^2 + \mathcal{R}(\mathbf{O}_n, \mathbf{O}_o). \tag{9}$$

In this way, we try to preserve the old task performance achieved at layer $l = L_o$ in the subsequent layers as well. In other words, by using the mapping $\bar{\mathbf{O}}$, we can provide the previous layer estimate of the target according to (6). Detailed steps of the proposed method is outlined in Algorithm 1.

In both of the above optimization problems, instead of using a simple weight decay as the regularization $\mathcal{R}$, we propose to use a weighted sum of ℓ_2-norm of $\mathbf{O}_n$ and $\mathbf{O}_o$ which suits the convex framework of SSFN. In particular, we define

$$\mathcal{R}(\mathbf{O}_n, \mathbf{O}_o) = \eta(\|\mathbf{O}_o\|_F^2 + \lambda \|\mathbf{O}_n\|_F^2), \tag{10}$$

where the regularization hyperparameter η requires cross-validation. This choice of regularization provides a more interpretable understanding of the bevahior of the network with respect to λ. In this way, we can find a closed-form expression for the solution of (8) as follows

$$\mathbf{O}_{ln}^{\star} = \mathbf{T}_n \mathbf{Y}_{ln}' (\mathbf{Y}_{ln} \mathbf{Y}_{ln}' + \eta\lambda \mathbf{I})^{-1} \tag{11}$$

$$\mathbf{O}_{lo}^{\star} = \mathbf{O}_{lo}^{\star} \mathbf{Y}_{ln} \mathbf{Y}_{ln}' (\mathbf{Y}_{ln} \mathbf{Y}_{ln}' + \frac{\eta}{\lambda} \mathbf{I})^{-1}. \tag{12}$$

It can be easily seen that when $\lambda \to 0$, we have $\mathbf{O}_{lo}^{\star} = 0$ and $\mathbf{O}_{ln}^{\star} = \mathbf{T}_n \mathbf{Y}_{ln}' (\mathbf{Y}_{ln} \mathbf{Y}_{ln}')^{-1}$; meaning that we will exactly achieve the maximum new task performance. On the other hand, when $\lambda \to +\infty$, we have $\mathbf{O}_{lo}^{\star} = \mathbf{O}_{lo}^{\star}$ and $\mathbf{O}_{ln}^{\star} = 0$, and we will exactly achieve the maximum old task performance as in before LwF. Similar expressions can be derived for the optimization problem (9) as well.

Algorithm 1 : Adaptive Learning without Forgetting

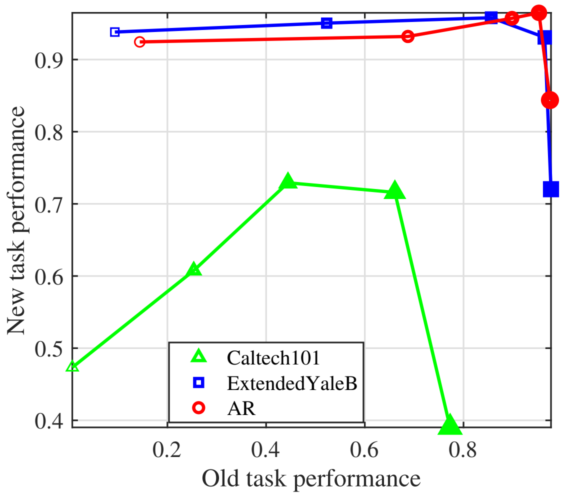
```
Input:
 1: ${\mathbf{O}_{lo}^{\star}}_{l=0}^{L_o}$: set of optimized weights on the old task for a network with $L_o$ layers
 2: $\mathbf{X}_n$, $\mathbf{T}_n$: training data and target pairs on the new task
 3: $L_n$: number of layers of the network for the new task
 4: $Q$: total classes of new and old tasks $Q_o + Q_n$
Initialization:
 1: $l = 0$
 2: $\mathbf{Y}_{0n} = \mathbf{X}_n$                    (Feature matrix at layer $l = 0$)
Adaptive growth of layers:
 1: repeat
 2:     if $l \leq L_o$ then
 3:         Find $\mathbf{O}_{ln}^{\star}$ and $\mathbf{O}_{lo}^{\star}$ using (8)
 4:     else
 5:         Find $\mathbf{O}_{ln}^{\star}$ and $\mathbf{O}_{lo}^{\star}$ using (9)
 6:     end if
 7:     Construct $\mathbf{O}_l^{\star} = \begin{bmatrix} \mathbf{O}_{lo}^{\star} \\ \mathbf{O}_{ln}^{\star} \end{bmatrix}$
 8:     Form the weight matrix $\mathbf{W}_{l+1} = \begin{bmatrix} \mathbf{V}_Q \mathbf{O}_l^{\star} \\ \mathbf{R}_{l+1} \end{bmatrix}$
 9:     Compute $\mathbf{Y}_{(l+1)n} = \mathbf{g}(\mathbf{W}_{l+1} \mathbf{Y}_{ln})$
10:     $l \leftarrow l + 1$
11: until $l > L_n$
```

IV. Experimental Results

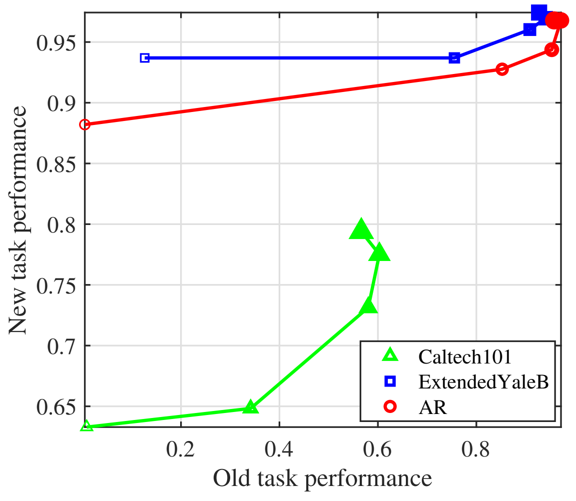
In this section, we carry out experiments to evaluate the effectiveness of the adaptive LwF scheme over SSFN architecture. We set the number of hidden neurons $n = 2Q + 1000$ in all the layers of the network. We report our results for three popular datasets in the literature in Table I. We divide each dataset into two parts: we randomly choose half of the classes as the old task and the other half as the new task. Note that we choose these datasets to have different similarity levels

TABLE I: Testing classification performance comparison over SSFN architecture $\lambda = 10$ for all datasets. We set $L_o = 5$ and $L_n = 10$ for adaptive LwF scheme. For LwF and joint training, we set $L = 10$.

Dataset	Joint Training			LwF			Adaptive LwF		
	old	new	both	old	new	both	old	new	both
AR	97.34	97.31	97.33	94.63	**97.53**	96.06	**95.36**	97.24	**96.29**
ExtendedYaleB	97.82	97.78	97.80	90.43	**97.93**	94.18	**93.04**	97.82	**95.43**
Caltech101	72.81	72.92	73.56	58.34	**76.15**	67.64	**60.67**	76.05	**68.85**



(a) New versus old task performance at layer $L = 0$



(b) New versus old task performance at layer $L = 10$

Fig. 1: Sensitivity of the network to λ in the first and last layer. A larger marker size indicates a larger value for λ while being changed over five values of 10^{-2}, 10^{-1}, 1, 10, 10^2.

between the old and new tasks in the LwF setup. AR dataset contains a set of face images of 50 males and 50 females. While the face images of different persons in ExtendedYaleB are less similar due to having an unbalanced number of males and females along with dissimilar illumination across classes. Caltech101 dataset is even more challenging as it contains dissimilar classes, e.g., animals, vehicles, flowers, etc.

In contrast to LwF [14], which uses multi-head evaluation, we report test classification accuracy (top-1 accuracy) by using single-head evaluation [10]. In multi-head evaluation, samples from two different tasks have no chance to get misclassified with one another. In other words, the task label of every sample is known as a priori and only the class label is assumed unknown. While in single-head evaluation, the task label is unknown as well resulting in a generally much harder problem. To account for the random partitioning of old and new datasets, we report the average test accuracy over 10 Monte Carlo simulation.

We carry out two sets of experiments. First, we implement adaptive LwF over SSFN where the number of layers changes from $L_o = 5$ for the old task to $L_n = 10$ for the new task. We compare this performance with that of LwF over SSFN with a fixed number of layers $L = 10$. We also report the performance of joint training with $L = 10$ layers on the old and new tasks as an upper bound benchmark. The test classification accuracy in percentage for the three datasets is reported in Table I. It can be seen that adaptive LwF consistently outperforms standard LwF on the old task and joint testing. The reason for this improvement is that choosing $L = 10$ layers for the old task leads to overfitting while extending the network from 5 to 10 layers can handle both tasks more accurately. At the same time, new task performance remains relatively unaffected resulting in higher overall performance. Note that the performance gap between adaptive LwF and joint training increases as we increase the difficulty of the datasets from AR to Caltech101.

In the second set of experiments, we investigate the sensitivity of the algorithm to the forgetting factor λ in Figure 1. We analyze the behavior of old and new task performance at layer $L = 0$ and $L = 10$ while changing λ over five values of 10^{-2}, 10^{-1}, 1, 10, 10^2. Larger marker size indicates a larger value of λ. It can be seen that new task performance is less sensitive to λ when we add more layers. In other words, new task performance can still be high on the last layer despite having low values in the first layer of the network. Therefore, the overall performance will be less sensitive to cross-validation on the parameter λ. This result suggests that deep neural networks might have a higher potential in the LwF scenario than a shallow network.

V. Conclusion

We showed that by using a layer-wise convex optimization approach, it is possible to develop an LwF algorithm that changes its size upon receiving the new dataset. The proposed neural network enjoys low-computational complexity due to the use of random weights and layer-wise training which makes it appealing for applications with limited resources. Generalizing this method for other neural network architectures, such as CNN and RNN, is a potential future direction.

References

[1] C. Tian, L. Fei, W. Zheng, Y. Xu, W. Zuo, and C.-W. Lin, "Deep learning on image denoising: An overview," *arXiv preprint arXiv:1912.13171*, 2019.

[2] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, Nov 2019.

[3] S. Chatterjee, A. M. Javid, M. Sadeghi, P. P. Mitra, and M. Skoglund, "Progressive learning for systematic design of large neural networks," *arXiv preprint arXiv:1710.08177*, 2017.

[4] A. Gepperth and B. Hammer, "Incremental learning algorithms and applications," in *European Symposium on Artificial Neural Networks (ESANN)*, Bruges, Belgium, 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01418129

[5] J. Xin, Z. Wang, L. Qu, and G. Wang, "Elastic extreme learning machine for big data classification," *Neurocomputing*, vol. 149, pp. 464–471, 2015.

[6] M. Wang and C. Wang, "Learning from adaptive neural dynamic surface control of strict-feedback systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 6, pp. 1247–1259, June 2015.

[7] S. Bianco, G. Ciocca, P. Napoletano, and R. Schettini, "An interactive tool for manual, semi-automatic and automatic video annotation," *Computer Vision and Image Understanding*, vol. 131, pp. 88 – 99, 2015, special section: Large Scale Data-Driven Evaluation in Computer Vision. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314214001544

[8] 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

[9] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," *Advances in Neural Information Processing Systems 30*, pp. 4652–4662, 2017. [Online]. Available: http://papers.nips.cc/paper/7051-overcoming-catastrophic-forgetting-by-incremental-moment-matching.pdf

[10] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr, "Riemannian walk for incremental learning: Understanding forgetting and intransigence," *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XI*, vol. 11215, pp. 556–572, 2018. [Online]. Available: https://doi.org/10.1007/978-3-030-01252-6_33

[11] S. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5533–5542, July 2017.

[12] G. Boukli Hacene, V. Gripon, N. Farrugia, M. Arzel, and M. Jezequel, "Transfer incremental learning using data augmentation," *Applied Sciences*, vol. 8, no. 12, 2018. [Online]. Available: https://www.mdpi.com/2076-3417/8/12/2512

[13] 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.

[14] 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.

[15] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," *Computer Vision – ECCV 2018*, pp. 144–161, 2018.

[16] P. Dhar, R. V. Singh, K. Peng, Z. Wu, and R. Chellappa, "Learning without memorizing," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5133–5141, June 2019.

[17] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, Jan 2018.

[18] S. Chatterjee, A. M. Javid, S. K. Mostafa Sadeghi, P. P. Mitra, and M. Skoglund, "SSFN: Self size-estimating feed-forward network and low complexity design," *arXiv preprint arXiv:1905.07111*, 2019.