IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 20, 2021, accepted September 17, 2021, date of publication September 27, 2021, date of current version October 12, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3115946

Learning Without Forgetting: A New Framework for Network Cyber Security Threat Detection

RUPESH RAJ KARN[1], PRABHAKAR KUDVA[2], (Senior Member, IEEE), AND IBRAHIM M. ELFADEL[1], (Senior Member, IEEE)

[1]Center for Cyber Physical Systems, Khalifa University, Abu Dhabi, United Arab Emirates
[2]IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA

Corresponding author: Ibrahim M. Elfadel (ibrahim.elfadel@ku.ac.ae)

This work was conducted under a Joint Study Agreement, W1463335, between IBM Research, Yorktown Heights, NY, USA, and Khalifa University, Abu Dhabi, UAE.

ABSTRACT Progressive learning addresses the problem of incrementally learning new tasks without compromising the prediction accuracy of previously learned tasks. In the context of artificial neural networks, several algorithms exist for achieving the progressive learning goal of *learning without forgetting*. However, these algorithms have traditionally been tested on the well-known and widely available datasets from the domain of image understanding and computer vision. Very little has been done on exploring the suitability of progressive learning algorithms in the important area of network threat detection. On a more fundamental level, progressive learning algorithms are still faced with the challenge of predicting the ultimate ability of a given neural network architecture to add more tasks to its repertoire without undergoing *catastrophic forgetting*. The goal of this paper is to address such a challenge in the context of cyber security threat detection. It does so by providing a unified conceptual and computational framework where progressive learning algorithms can be analyzed, compared, and contrasted in terms of their learning capacity and prediction accuracy for specific datasets from the cloud cyber security domain. In particular, this paper provides rigorous metrics for predicting the onset of catastrophic forgetting in the cyber security domain and contrasts them with their usage in the imaging domain. Our extensive numerical results show that progressive learning, along with the proposed criteria for catastrophic forgetting, provides a very structured framework for automating network threat detection as new threats emerge throughout network operation.

INDEX TERMS Progressive learning, neural network, synaptic intelligence, elastic weight consolidation, fisher information, Hessian matrix, matrix rank, cyber security, threat detection.

I. INTRODUCTION

Progressive learning addresses the problem of incrementally learning new tasks without compromising the prediction accuracy of previously learned tasks. In the context of feed-forward artificial neural networks (ANN), there are two major approaches to the embodiment of progressive learning. The first is biologically-inspired and consists of *growing* the network and therefore augmenting its parameter space to accommodate the new tasks. Examples of such approach include [1] as well as the body of learning algorithms based on architectural ANN search that fall under the *AutoML* paradigm [2], [3]. The second approach assumes that the ANN architecture is fixed and focuses on ANN weight management to achieve its goals. Specifically, the synaptic weights that are important for generalizing the older tasks are preserved with the equal-loss surfaces in the weight space changed minimally during the learning of a new task. Such algorithms include Synaptic Intelligence (SI) [4], Elastic Weight Consolidation (EWC) [5], Orthogonal Weight Modification [6], continual learning with the Kalman optimizer (CLKO) [7], Gradient Episodic Memory [8], Memory Aware Synapses [9], and Random Path Selection [10]. Generally, various types of regularizations are applied to preserve the important weights of the neural network model. Such regularization framework enables the same model to generalize novel as well as older tasks with minimal impact on the generalization accuracies of the older tasks. This accuracy balance between older and newer tasks may be construed a manifestation of the stability-plasticity dilemma of biological

The associate editor coordinating the review of this manuscript and approving it for publication was Rongbo Zhu.

neural networks. This dilemma refers to the ability of the neural system to integrate new knowledge while internally compensating to prevent the loss of old knowledge. Such ability is foundational to continual lifelong learning [11].

To date, most of the progressive learning algorithms have been illustrated and tested on imaging datasets such as MNIST [12] and CIFAR [13]. As is now traditional in image recognition, convolutional neural networks (CNN) have been used in feed-forward architectures upon which the progressive learning algorithms have been implemented. Successive CNN layers are used to identify simple features in the data, which are then aggregated to create the complex features used in the higher decision layers for image classification. In such a process, due to translation invariance of the convolution filter, feature location within the image is not of high relevance. This is the case not just for spatial data but also for temporal data where translation-invariant filters are used to detect salient features, as is the case in audio processing, natural language processing, or sensor waveform processing. On the other hand, there are application domains where the location of the feature within the data is of very high relevance. One such domain is that of network data collected for network security monitoring. While imaging datasets have pixels activating all the input nodes of a CNN, in a cloud security application, the input nodes of an ANN are activated with signals, each having its own physical interpretation in the cloud domain. Under such conditions, progressive models that perform efficiently in image recognition are not guaranteed to perform as well in the cloud cyber security domain.

Yet, the latter domain is a very natural context for progressive learning as cloud monitoring signals often expose new types of cyber security attacks. Indeed, a threat detection model trained for a group of attacks such as *worm, Dos, shellcode, and exploits* [14] has to learn new types of attacks such as *backdoor and fuzzers*. Each group of attack labels is a learning *task*, and so one training task consists of learning *worm, Dos, shellcode, and exploits*, while another learning task consists of learning *backdoor and fuzzers*. A major theme of this paper is to explore the use of progressive learning in the cloud cyber security domain. Another major theme of this paper is to address a fundamental problem of progressive learning, which is the fact that for a given neural architecture, learning cannot go forever.

Indeed, as new tasks are added to the ANN learning repertoire, it comes to a critical point where the ANN runs a real risk of losing the knowledge of previously learned tasks. This risk is manifested in a phenomenon known sd *catastrophic forgetting* [15], which may be defined as the sudden loss of generalization accuracy based on the previously learned tasks. This loss occurs because the ANN weights responsible for the exact generalization based on the old tasks are compromised as they are modified to adapt to the new tasks. Several attempts have been made to develop an algorithm that mitigates catastrophic forgetting. The mitigation algorithms include regularization, sparse encoding, dual memory, rehearsal, ensemble [15], generative replay [16], reinforced learning [17], and phantom sampling [18]. In spite of these mitigation attempts, catastrophic forgetting remains an open problem, and current solutions to measure it do not adequately estimate the ''saturation'' level in the weight space of the ANN *prior* to learning additional tasks. An attempt has been made to measure the catastrophic forgetting through several metrics in [15]. But these metrics are based on measurements bearing on the datasets and the cross-validation accuracies of various trained tasks, and do not account for the ANN synaptic weights as they shift between various tasks. Metrics based on weight behavior are essential to assess whether the ANN is capable of learning new tasks or not. As pointed out in [15], there is a pressing need for such metrics, which should apply to a wide range of progressive learning algorithms. One of the major contributions of this paper is to propose such metrics, both exact and heuristic, and validate them on a variety of progressive learning algorithms using cyber security datasets.

II. CONTRIBUTIONS AND ORGANIZATION

As already mentioned, in progressive learning based on feed-forward neural networks, the algorithm attempts to preserve the equal-loss contours of weights that are *important* for old tasks while tuning other available weights during back-propagation for adapting them as precisely as possible to the new tasks. It is intuitively clear that after training a large number of progressive tasks, most of the ANN weights will become ''important,'' and as a result, the dual goals of preserving the equal-loss contours of the old tasks and tuning the network to adapt to the new tasks become inherently conflicting. In this paper, we have used the term ''congestion'' to describe the competition between older and newer tasks for access to ANN weights. Not only does such congestion cause the catastrophic forgetting of previously learned tasks, but also it negatively impacts the learning accuracy of new tasks.

We are now in a position to summarize the key contributions of this paper:

1) We provide a unified conceptual and computational framework for the comparative assessment of three major progressive learning algorithms: Synaptic Intelligence (SI) [4], Elastic Weight Consolidation (EWC) [5], and Orthogonal Weight Modification (OWM). [6].
2) We extend the application domain of progressive learning from imaging to cyber security and illustrate the advantages that such a learning paradigm can achieve on network threat detection.
3) We use the unified computational framework to introduce, evaluate, and compare exact and heuristic criteria for predicting the onset of catastrophic forgetting and network congestion. We further contrast these criteria according to the two application domains of imaging and cyber security.
4) We provide algorithms for accelerating the computation of the second-order information used in the exact rules

for congestion prediction. We implement these algorithms for classification loss functions.

5) We use the unified conceptual and computational framework to introduce a new progressive learning algorithm called Fisher Synaptic Intelligence (FSI) that integrates the probabilistic features of EWC with the task specifications of SI. The new FSI algorithm is implemented and compared with both SI and EWC from the viewpoint of network congestion on both cyber security and imaging datasets.
6) Finally, we provide a computational analysis of label grouping amongst various tasks and evaluate its impact on network congestion. This analysis is generic and applies to all the progressive learning algorithms considered in this paper.

The remainder of this paper is organized as follows. In Section III, the three major progressive learning algorithms of SI, EWC, and OWM are surveyed under a unified conceptual and mathematical framework. In Section IV, exact and heuristic metrics for predicting the onset of network congestion are introduced. The exact metric is based on the evaluation of the rank of the Hessian matrix of the loss function. Also in Section IV, methods for the fast estimation of the Hessian rank for cross-entropy loss functions are presented. In Section V, the unified computational framework for comparing the various progressive learning algorithms is described, along with the network security datasets UNSW and AWID. The extensive numerical experiments for each of the three progressive learning algorithms are presented in Section VI, along with an evaluation of catastrophic forgetting for each case. The new FSI algorithm is introduced and evaluated in Section VII. Section VIII is devoted to design issues such as the impact of label grouping and weight sharing on learning without forgetting. Also in Section VIII, the two application domains of network security and imaging are compared and contrasted. Finally, the paper is concluded in Section IX where recommendations for future work are also provided. Additional results and figures illustrating the various points made in this work have been assembled in an appendix attached to the end of the article.

This article is a major expansion of our earlier conference publication [19], where the focus was only on the SI algorithm and the least-squares loss function, which corresponds in part to Subsections III-A and VI-A of this article. The rest of the article is entirely original. To facilitate its reading, the reader is encouraged to refer to Table 1, which has the definitions of the various abbreviations used throughout the article.

TABLE 1. List of abbreviations.

Abbreviation	Description
ANN	Artificial Neural Network
AWID	Aegea Wi-Fi Intrusion Dataset
CIFAR	Canadian Institute for Advanced Research
CLKO	Continual Learning with the Kalman Optimizer
CNN	Convolutional Neural Network
EWC	Elastic Weight Consolidation
FSI	Fisher Synaptic Intelligence
LM	Levenberg-Marquardt
MNIST	Modified National Institute of Standards and Technology
OWM	Orthogonal Weight Modification
SI	Synaptic Intelligence
UNSW	University of New South Wales

III. PROGRESSIVE LEARNING: UNIFIED FRAMEWORK

In this section, we describe a unified conceptual framework for presenting three progressive learning algorithms, namely, Synaptic Intelligence (SI) [4], Elastic Weight Consolidation (EWC) [5], and Orthogonal Weight Modification (OWM) [6]. All these algorithms belong to the supervised learning paradigm and use one form or another of gradient descent to incrementally find the optimal ANN weights as more tasks are presented to the network for training.

A. SYNAPTIC INTELLIGENCE

Synaptic Intelligence [4] is based on the premise that progressive learning can proceed in the parameter space $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_P) \in \mathbb{R}^P$ if there is a subset of the P parameters that is available for tuning to accommodate the new task while the complementary subset is held unchanged to account for the learning of the prior tasks. Two important concepts are used to translate this intuitive idea into an algorithm: proxy loss and weight importance. To explain these two important concepts, assume that the ANN has learned μ tasks by optimizing a loss function $\tilde{L}_\mu(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. To learn the $\mu + 1$ task without compromising the past learning of the μ prior tasks, a proxy loss function $\tilde{L}_{\mu+1}(\boldsymbol{\theta})$ is introduced:

$$\begin{aligned}\tilde{L}_{\mu+1}(\boldsymbol{\theta}) &= L_{\mu+1}(\boldsymbol{\theta}) + cL_{\tau\leq\mu}(\boldsymbol{\theta}) \\ &= L_{\mu+1}(\boldsymbol{\theta}) + c\sum_{k=1}^{P}\Omega_k^\mu(\theta_k^\mu - \theta_k)^2 \end{aligned} \quad (1)$$

where τ denotes a task and $L_{\mu+1}(\boldsymbol{\theta})$ is the actual loss of the $(\mu+1)th$ task only, which can be the standard least-squares or the cross-entropy loss function. The term $\Omega_k^\mu(\theta_k^\mu - \theta_k)^2$ is the surrogate loss which approximates the loss function of the previous tasks $L_{\tau\leq\mu}(\boldsymbol{\theta})$ in the neighborhood of $\boldsymbol{\theta}^\mu$. During the back-propagation phase of training, the model considers the loss with respect to the older tasks through the surrogate loss component, $L_{\tau\leq\mu}$, along with the current task's loss function $L_{\mu+1}$. The symbol $\boldsymbol{\theta}^\mu = (\theta_1^\mu, \theta_2^\mu, \ldots, \theta_P^\mu)$ denotes the vector of optimized learning parameters of the first μ tasks, and Ω_k^μ denotes the *importance* of the k-th ANN parameter to the loss function of the prior μ tasks. The higher this importance coefficient, the less incentive there is to change θ_k from its μ-optimal value θ_k^μ. The role of the additional quadratic summation in (1) is to enforce this proximity according to the importance of each ANN parameter. The c coefficient in (1) is used to balance the need for parameter update as required by the regular loss function $L_{\mu+1}(\boldsymbol{\theta})$ against the proximity penalty $L_{\tau\leq\mu}(\boldsymbol{\theta})$ in order to preserve previously learned tasks. The expression of

the importance of k-th ANN parameter is given by

$$\Omega_k^{\mu} = -\sum_{\tau \le \mu} \frac{\partial \tilde{L}_{\tau}(\boldsymbol{\theta})}{\partial \theta_k} \frac{\Delta_k^{\tau}}{(\Delta_k^{\tau})^2 + \epsilon} \tag{2}$$

$$\Delta_k^{\tau} = \theta_k^{\tau} - \theta_k \tag{3}$$

The interpretation of the importance coefficient of an ANN parameter θ_k is that it is proportional to both the rate of variation of the loss function with respect to θ_k and to the actual change Δ_k^{τ} of $\boldsymbol{\theta}_k$ with respect to its starting value. The normalizing term $(\Delta_k^{\tau})^2 + \epsilon$ is used to scale each term in the proxy loss summation while ϵ is used to avoid dividing by zero in case of zero change in the k-th parameter.

B. ELASTIC WEIGHT CONSOLIDATION

As in the Synaptic Intelligence (SI) method previously described, the Elastic Weight Consolidation (EWC) algorithm is based on the principle of having the optimizer remember the old tasks by selectively slowing down the updates on the weights that are important for those tasks. To better understand this statement, we adopt the terminology and concepts of the original EWC publication [5].

EWC is based on the premise that there is a weight vector $\boldsymbol{\theta}^{\mu+1}$ for accurately learning task $\tau = \mu + 1$ that is close to the previously found weight vector $\boldsymbol{\theta}^{\mu}$ for tasks $\tau \le \mu$. When learning $\tau = \mu + 1$, EWC therefore protects the accuracy in $\tau \le \mu$ by constraining the weights to stay in a region of low error centered around $\boldsymbol{\theta}^{\mu}$. To define the weights that are most important for a given task, EWC uses a probabilistic framework for neural network training. Let D_{τ} be the dataset corresponding to task τ, and let $p(D_{\tau}|\theta^{\tau})$ be the probability density function of the D_{τ} given the model parameters θ^{τ}. Then the Fisher's information matrix of the data D_{τ} is given as the ensemble average

$$\mathbf{F}(\boldsymbol{\theta}_{\boldsymbol{\tau}}) = \mathbf{E}\left[(\nabla_{\boldsymbol{\theta}} \log p(D_{\tau}|\boldsymbol{\theta}^{\tau}))(\nabla_{\boldsymbol{\theta}} \log p(D_{\tau}|\boldsymbol{\theta}^{\tau}))^T\right] \tag{4}$$

where ∇_{θ} designate the row gradient vector with respect θ. The Fisher's information matrix for task τ can also be expressed in terms of the likelihood function $p(\theta|D_{\tau})$ in which case the EWC loss function $\tilde{L}_{\mu+1}$ for training task $\tau = \mu + 1$ is expressed as

$$\tilde{L}_{\mu+1}(\boldsymbol{\theta}) = L_{\mu+1}(\boldsymbol{\theta}) + \frac{\beta}{2} \sum_{k=1}^{P} F_k^{\mu} (\theta_k^{\mu} - \theta_k)^2 \tag{5}$$

where F_k^{μ} is the k-th diagonal coefficient of the Fisher's information matrix (4) for the μ-th task. The component $L_{\mu+1}(\boldsymbol{\theta})$ is the loss for task $\tau = \mu + 1$ only, β is a penalty coefficient penalizing the deviation from optimal parameters θ^{μ} of task μ. It plays a role similar to that of c in (1). Note that the loss function $L_{\mu+1}(\boldsymbol{\theta})$ may be a least-squares or a cross-entropy loss function. To compute the Fisher's diagonal coefficient for task μ, a statistical average is used for the Fisher's information matrix according to the formula

$$\mathbf{F}^{\mu}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{m=1}^{N} \left[\nabla_{\boldsymbol{\theta}} \log p(D_{\mu}^{(m)}|\boldsymbol{\theta}^{\boldsymbol{\mu}})\right] \times \left[\nabla_{\theta} \log p(D_{\mu}^{(m)}|\boldsymbol{\theta}^{\boldsymbol{\mu}})\right]^T \tag{6}$$

where $D_{\mu}^{(m)}$ is the m-th sample of the data D_{μ} for task μ, and N is the number of samples. For the Fisher's diagonal coefficient in (5), the above statistical average is specialized to

$$F_k^{\mu} = \frac{1}{N} \sum_{m=1}^{N} \left[\frac{\partial}{\partial \theta_k} \log p(D_{\mu}^{(m)}|\boldsymbol{\theta}^{\boldsymbol{\mu}})\right]^2 \tag{7}$$

In the EWC [5] algorithm implementation at [20], the D_{τ} training data are considered independent of each other, and the surrogate loss $F_k^{\mu}(\theta_k^{\mu} - \theta_k)^2$ in (5) depends only on task μ rather than tasks $\tau \le \mu$ as in (1). This aspect will be addressed further in Section VI-A.

Finally, note that the Fisher's coefficient F_k^{μ} in EWC (5) plays a role similar to that of Ω_k^{μ} in SI (1).

C. ORTHOGONAL WEIGHT MODIFICATION

Assume that the training data for task τ has N samples denoted by $D_{\tau} = [D_{x_{\tau}}, D_{y_{\tau}}]$ where D_x's are training samples and D_y's are labels.

The Orthogonal Weight Modification (OWM) [6] algorithm is based on the principle that the model remembers the old tasks by modifying its weights only in the direction orthogonal to the subspace spanned by the training data of the old. Unlike SI and EWC, the OWM algorithm does not use a surrogate loss function to remember older tasks. Rather, the standard gradient descent function is scaled with an orthogonal projection transformation to update the weights considering that an orthogonal weight update does not interfere with older weights. To better explain this statement, we adopt the terminology of [6]. We first assume that the network has $l+1$ layers indexed by $l = 0, 1, 2, \ldots, L, L+1$, with $l = 0$ and $l = L + 1$ being the input and output layers, respectively. Assume the network has already learned μ tasks by optimizing a loss function $L_{\mu}(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. To learn the $(\mu+1)$-st task without compromising the past learning of the μ prior tasks, the standard ANN loss function $L_{\mu+1}(\boldsymbol{\theta})$ is minimized by updating weights according to the gradient descent

$$G(\boldsymbol{\theta}) = \left[\nabla_{\theta} L_{\mu+1}(\boldsymbol{\theta})\right]^T \tag{8}$$

$$\boldsymbol{\theta} \rightarrow \boldsymbol{\theta} - \kappa \mathbf{O}_{\mu+1} G(\boldsymbol{\theta}) \tag{9}$$

$$\mathbf{O}_{\mu+1} = \mathbf{I} - D_{\tau \le \mu}(D_{\tau \le \mu}^T D_{\tau \le \mu})^{-1} D_{\tau \le \mu} \tag{10}$$

where $D_{\tau \le \mu}$ designates the data matrix of all tasks $\tau \le \mu$, κ is the learning rate, $\mathbf{I}$ is the identity matrix, and $\mathbf{O}_{\mu+1}$ is a projection matrix whose columns generate a subspace that is orthogonal to the subspace generated by $D_{\tau \le \mu}$. Indeed, it is easy to verify that $D_{\tau \le \mu} \mathbf{O}_{\mu+1} = \mathbf{0}$. In (9), the projection matrix is constructed to be conformal to the dimension P of the parameter space.

The recommended method to compute $\mathbf{O}_{\mu+1}$ is through an iterative process as in the Recursive Least Squares algorithm [6], with $\mathbf{O}_{\mu+1}$ computed for each layer separately using only the training dataset $D_{\mu+1}$ and the projection for the most recent task $\mathbf{O}_{\mu}$.

TABLE 2. Comparison of SI, EWC, and OWM.

Feature	SI	EWC	OWM
Proxy loss	✓	✓	×
Importance coefficients	✓	✓	×
Penalty parameter	✓	✓	×
Weight updates	Gradient	Gradient	Projected gradient
Vanishing gradient	✓	×	✓
Catastrophic forgetting	✓	✓	✓
Congestion metrics	All	All	Only heuristics

The OWM formulation is different from that of SI and EWC as it abides by the traditional back-propagation algorithms and does not use any proxy loss functions or any importance coefficients such as Ω_k^μ in SI (2) and F_k^μ in EWC (7).

D. COMPARISONS

A summary of the features of the three progressive learning algorithms SI, EWC, and OWM is provided in Table 2. Note that EWC has the distinct advantage of being less prone to the vanishing gradient problem. This problem is well known in ANN training using gradient descent and backpropagation [21], and results in certain weights failing to get updated when their loss function partial derivatives are close to zero. This problem is directly inherited by the projected gradient of OWM (8). In the context of SI progressive learning, the vanishing gradient will also impact the importance parameters of Eq. (2) and hamper their use as weights in the proxy loss functions. On the other hand, the Fisher coefficients of the EWC algorithm are less likely to be impacted by the vanishing gradient problem as a result of the statistical averaging of (6). All these algorithms are prone to catastrophic forgetting, with only heuristic methods amenable to be used in OWM for its prediction. Details will be given in the next section.

IV. LEARNING WITHOUT FORGETTING AND ITS METRICS

A natural barrier to continuous progressive learning in fixed ANN architectures is of course the pre-fixed, finite number of neurons in the ANN architecture. Network congestion will occur when the network capacity to learn new tasks without compromising the accuracy of prior tasks is reached. A symptom of network congestion is the conflict between older and new tasks for weight tuning and the subsequent loss of accuracy due to conflicting demands on weight updates. The main goal of the fixed-architecture algorithms presented in the previous section is to delay the occurrence of network congestion as much as possible by managing the weight space and scheduling weight updates according to accuracy requirements across all tasks. In this section, we present various quantitative metrics for detecting network congestion during progressive learning.

A. WEIGHT IMPORTANCE

In all the progressive learning algorithms described in Section III, a restriction criterion is typically applied over the trajectory of ''important'' weights so that their paths do not deviate much with respect to those defined by the older tasks [4]–[6]. The model undergoes congestion when most of the weights become ''important'' as far as the older tasks are concerned. It follows that inspecting the number of important weights iteratively after training every task can signal the occurrence of congestion. Intuitively, the number of important weights should increase as the model learns a new task. The model is on the verge of congestion when it approaches the number of weights P in the ANN. Beyond P, the number of important weights starts to decrease in the numerical experiments of Section VI. The intuition behind such decline is that once the network is congested, the importance coefficients become dependent on the data of several learned tasks at once. The contributions of these tasks to a given importance parameter may cancel out, which could result in decreasing the algebraic value of the importance coefficient. As discussed in Section III, SI uses Ω_k^μ defined in (2) while EWC uses the F_k^μ defined in (7) to measure weight importance. The value of importance can be either positive or negative. A weight can be treated as non-important if its importance value is zero. In our detection methodology, we counted the number of non-zero importance values for SI and EWC to determine weight space utilization and denoted them as $n(\Omega)$ and $n(F)$, respectively.

B. HESSIAN MATRIX RANK

The ANN should become *congested* as soon as the size of the set of important weights is too high relative to the total number of weights. The extensive numerical examples of Section VI will show that it is possible to effectively detect the ANN congestion using importance coefficients. However, such simple congestion metric has three main shortcomings:

1) The lack of a clear mapping between weight subsets and learned tasks in a given ANN. This often results from the interplay between the backpropagation algorithm, the ANN connectivity graph, and the weights initialization.
2) Correlated tasks often result in weight sharing, which lowers the threshold at which congestion starts to occur. Due to task correlation, the number of important weights may decrease with the training of new tasks without showing congestion.
3) The use of the importance metric is restricted to progressive learning algorithms such as SI or EWC, where a proxy loss function is used. It cannot be used with algorithms such as OWM where the notion of weight importance is not defined.

These shortcomings can be overcome if we use the second-order information provided by the hyper-surface of the ANN cost function. This second-order information is the Hessian matrix evaluated at the optimal weights where the ANN cost function has a local minimum for the set of tasks already learned.

The change in the loss function of a model that has learned μ tasks can be approximated in the neighborhood of $\boldsymbol{\theta}^\mu$ by

$$\Delta\tilde{L}_\mu(\boldsymbol{\theta}) \approx \Delta\boldsymbol{\theta}^T \boldsymbol{H}(\boldsymbol{\theta}^\mu)\Delta\boldsymbol{\theta} \tag{11}$$

where $\Delta\boldsymbol{\theta} = \boldsymbol{\theta} - \boldsymbol{\theta}^{\mu}$ and $\boldsymbol{H}(\boldsymbol{\theta}^{\mu})$ is the $P \times P$ Hessian matrix of the loss function computed at $\boldsymbol{\theta}^{\mu}$. To simplify the expressions, we use the notation $\boldsymbol{H}^{\mu} = \boldsymbol{H}(\boldsymbol{\theta}^{\mu})$. The (i, j) element of the Hessian matrix, denoted by h_{ij}^{μ}, is the second-order derivative of the loss function and is given by

$$h_{ij}^{\mu} = \frac{\partial^2 \tilde{L}_{\mu}(\boldsymbol{\theta}^{\mu})}{\partial\theta_i \partial\theta_j}$$

It is well known that the Hessian matrix is symmetric, and when computed at the point where the loss function has a local minimum, it is positive semi-definite. We propose to use the rank R of the Hessian matrix as a measure of network congestion after the learning of μ tasks. The rank R is a function of the number of learned tasks, i.e., $R = R(\mu)$, which increases with μ and is bounded by P. When $R(\mu) = P$, the ANN is fully congested, and learning without forgetting is impossible. In our conference paper [19], we showed the relevance of rank to congestion using the spectral decomposition of the Hessian matrix and illustrated with extensive numerical experiments using the SI algorithm. In the following paragraphs, we give further theoretical justification for the use of the rank of the loss function Hessian matrix to detect congestion.

Assume the network has been trained for task μ using the dataset D_{μ} and that the optimal weight found is $\boldsymbol{\theta}^{\mu}$. A new task $\tau = \mu + 1$ arrives for training. The loss function in the neighborhood of $\boldsymbol{\theta}^{\mu}$ can be approximated as a second-order Taylor series expansion[1]

$$\tilde{L}(\boldsymbol{\theta}, D_{\mu}) = \tilde{L}(\boldsymbol{\theta}^{\mu}, D_{\mu}) + \frac{1}{2}\Delta\boldsymbol{\theta}^T H(\boldsymbol{\theta}^{\mu}, D_{\mu})\Delta\boldsymbol{\theta}$$

where the first-order term, $\Delta\boldsymbol{\theta}^T G(\boldsymbol{\theta}^{\mu}, D_{\mu})$ vanishes at the optimal weight. The notation $G(\boldsymbol{\theta}^{\mu}, D_{\mu}) = \left[\nabla_{\boldsymbol{\theta}} \tilde{L}(\boldsymbol{\theta}^{\mu}, D_{\mu})\right]^T$ designates the gradient. When the network is being trained for task $\tau = \mu + 1$ using the dataset $D_{\mu+1}$, the loss function in the neighborhood of $\boldsymbol{\theta}^{\mu}$ is written as:

$$\tilde{L}(\boldsymbol{\theta}, D_{\mu+1}) = \tilde{L}(\boldsymbol{\theta}^{\mu}, D_{\mu+1}) + \Delta\boldsymbol{\theta}^T G(\boldsymbol{\theta}^{\mu}, D_{\mu+1}) + \frac{1}{2}\Delta\boldsymbol{\theta}^T H(\boldsymbol{\theta}^{\mu}, D_{\mu+1})\Delta\boldsymbol{\theta}$$

Note that the data dependence in the previous equation is on the new dataset $D_{\mu+1}$ for which the gradient $G(\boldsymbol{\theta}^{\mu}, D_{\mu+1})$ does not vanish as $\boldsymbol{\theta}^{\mu}$ is no longer optimal for the new task $\mu+1$. One possible approach for training the network to account for the new task $\tau = \mu + 1$ is to find the weight update $\Delta\boldsymbol{\theta}$ according to the constrained optimization problem:

$$\min_{\Delta\boldsymbol{\theta}}\{\frac{1}{2}\Delta\boldsymbol{\theta}^T H(\boldsymbol{\theta}^{\mu}, D_{\mu+1})\Delta\boldsymbol{\theta}\}$$

such that

$$\Delta\boldsymbol{\theta}^T G(\boldsymbol{\theta}^{\mu}, D_{\mu+1}) = 0.$$

The minimization problem is formulated with the Hessian matrix corresponding to $\tau = \mu$, which is meant to preserve the contours of the weights for all the older tasks $\tau \leq \mu$ and therefore prevent catastrophic forgetting. The constraint using the loss function gradient is meant to achieve the necessary optimality condition for $\tau = \mu + 1$ along the direction of the updated weight. Using the scalar Lagrange multiplier υ corresponding to the equality constraint, one can derive the necessary condition satisfied by the weight update $\Delta\boldsymbol{\theta}$

$$H(\boldsymbol{\theta}^{\mu}, D_{\mu+1})\Delta\boldsymbol{\theta} + \upsilon G(\boldsymbol{\theta}^{\mu}, D_{\mu+1}) = 0$$

Multiplying the above equation by $\Delta\boldsymbol{\theta}^T$, one gets

$$\Delta\boldsymbol{\theta}^T H(\boldsymbol{\theta}^{\mu}, D_{\mu+1})\Delta\boldsymbol{\theta} = 0$$

Since the Hessian at the optimal weight $\boldsymbol{\theta}^{\mu}$ is symmetric positive semi-definite, the above equation is satisfied by a non-zero $\Delta\boldsymbol{\theta}$ if and only if the Hessian matrix is rank-deficient.

When the Hessian matrix has full rank, the constrained minimization problem has no feasible solution, and the only solution possible is one that will involve some form of catastrophic forgetting.

C. HESSIAN MATRIX APPROXIMATION

In [19], we explored two techniques for computing the elements of the Hessian matrix. The first one is exact and is based on the second-order finite difference formula, and the second is an approximation based on the sum-of-squares format of the ANN loss function. The exact calculation is simple to implement but excessively time-consuming [19] while the approximation is more efficient as it requires only first-order information for computing the Hessian matrix. In this section, we derive a Hessian approximation expression for the entropy loss function, which is the more common loss function in classification problems.

Consider a current task τ with N training samples having $\boldsymbol{x}_m$, $1 \leq m \leq N$, as the input features, and y_m, $1 \leq m \leq N$, as the labels. The input-output mapping from the features $\boldsymbol{x}_m$ to the label y_m is denoted as $\tilde{\Gamma}_{\tau}(\boldsymbol{\theta}, \boldsymbol{x}_m)$. The (i, j) Hessian element for task τ is

$$h_{ij}^{\tau} = \frac{\partial^2 \tilde{L}_{\tau}(\boldsymbol{\theta})}{\partial\theta_i \partial\theta_j}$$

where $\tilde{L}_{\tau}(\boldsymbol{\theta})$ is given by the cross-entropy loss function

$$\tilde{L}_{\tau}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{m=1}^{N} y_m \log[\tilde{\Gamma}_{\tau}(\boldsymbol{\theta}, \boldsymbol{x}_m)] - \frac{1}{N}\sum_{m=1}^{N}(1 - y_m)\log[1 - \tilde{\Gamma}_{\tau}(\boldsymbol{\theta}, \boldsymbol{x}_m)] \quad (12)$$

To get more insight into above equation, consider a binary classification between labels $\{0, 1\}$. Note that the terms of the entropy loss functions are very small when the classification is correct, i.e., $\tilde{\Gamma}_{\tau}(\boldsymbol{\theta}, \boldsymbol{x}_m) \approx y_m$ and becomes large when $\tilde{\Gamma}_{\tau}(\boldsymbol{\theta}, \boldsymbol{x}_m)$ deviates significantly from y_m.

[1]For clarity, we make the dependence on the training data explicit.

To approximate the Hessian of the entropy loss function, the gradient is first computed:

$$\frac{\partial \tilde{L}_\tau(\boldsymbol{\theta})}{\partial \theta_i} = -\frac{1}{N}\sum_{m=1}^{N} \frac{y_m}{\tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)} \frac{\partial \tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)}{\partial \theta_i} + \frac{1}{N}\sum_{m=1}^{N} \frac{1-y_m}{1-\tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)} \frac{\partial \tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)}{\partial \theta_i}$$

To calculate the Hessian, we take the derivative once more

$$\begin{aligned}\frac{\partial^2 \tilde{L}_\tau(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j} =& -\frac{1}{N}\sum_{m=1}^{N} \frac{y_m}{\tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)} \frac{\partial^2 \tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)}{\partial \theta_i \partial \theta_j} \\ &+ \frac{1}{N}\sum_{m=1}^{N} \frac{y_m}{[\tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)]^2} \frac{\partial \tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)}{\partial \theta_j} \frac{\partial \tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)}{\partial \theta_i} \\ &+ \frac{1}{N}\sum_{m=1}^{N} \frac{1-y_m}{1-\tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)} \frac{\partial^2 \tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)}{\partial \theta_i \partial \theta_j} \\ &- \frac{1}{N}\sum_{m=1}^{N} \frac{1-y_m}{[1-\tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)]^2} \frac{\partial \tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)}{\partial \theta_j} \frac{\partial \tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)}{\partial \theta_i}\end{aligned}$$

When the output $\tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m) \approx \boldsymbol{y}_m$, the first and third terms in the above expression cancel out, while the second and fourth terms are simplified, and we get for the (i, j) Hessian element the approximate expression

$$h_{ij}^\tau \approx \frac{1}{N}\sum_{m=1}^{N} \left[\frac{1}{\tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)} - \frac{1}{1-\tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)}\right] \times \frac{\partial \tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)}{\partial \theta_i} \frac{\partial \tilde{\Gamma}_\tau(\boldsymbol{\theta}, \boldsymbol{x}_m)}{\partial \theta_j} \quad (13)$$

The above equation indicates that the Hessian can be approximated as the weighted outer products of the model gradients scaled with the difference of the model output reciprocals and averaged over all the training samples. This simplified Hessian expression, along with the fact that model gradients are already computed by the back-propagation training algorithm, results in a more efficient computation than the finite-difference Hessian formula [19]. In the experiments of this paper, the Hessian rank is computed based on the approximation of (13).

D. HEURISTIC METRICS

While the Hessian approximation (13) speeds up the calculation of the Hessian matrix, computing the Hessian rank is itself time-consuming as it has a cubic time complexity in the number of parameters P. It is therefore desirable to complement the weight importance and Hessian rank metrics with heuristic approaches that are less computationally demanding but nonetheless capable of providing rough predictions of neural network congestion. In a heuristic approach, a snapshot of the weights is taken after each progressive task training, and the "distance" between the weights of two successive tasks is measured. A large distance is an indicator of ANN congestion and of possible catastrophic forgetting. Such a large distance may occur if all the parameters of the ANN are important and change significantly to learn new tasks. Heuristic distance metrics are typically based on the notion of similarity between vectors in a linear space. Four metrics are proposed and evaluated for the measurement of similarities between weights. They are *Euclidean distance* [22], *Cosine angle* [23], *Jaccard similarity* [24], and *SequenceMatcher* [25]. The Euclidean and cosine metrics measure the quantitative deviation between two snapshots of weight vectors while the Jaccard and SequenceMatcher metrics measure qualitative similarity, i.e., the fraction of the total number of weights that remain unchanged between two successively learned tasks. The Euclidean distance measurement can attain any value between zero and infinity. A small Euclidean distance indicates that the two weight vectors are similar. On the other hand, the cosine, Jaccard, and SequenceMatcher metrics take values between 0 and 1, and a value close to 1 is an indication that the two weight vectors are similar.

E. COMPUTATIONAL COMPLEXITY

The computational cost associated with each of the congestion metrics proposed above is described in the following paragraphs.

1) WEIGHT IMPORTANCE

As indicated in Section IV-A, the number of non-zero importance values is used to evaluate weight space usage for SI and EWC. Because each weight parameter has an importance value, the size of the importance vector is equal to P, the dimension of the weight vector. According to Eq. (2), the computation of each importance parameter requires $\mathcal{O}(3\mu)$ multiplications, assuming that the computational cost of a floating-point square or division is the same as a floating-point multiplication. The computational complexity of the importance vector is therefore $\mathcal{O}(3\mu P)$.

2) APPROXIMATE HESSIAN RANK

The Hessian matrix elements are computed using Eq. (13). For each element, two multiplication and two division operations are required to obtain each term in the summation for a total of $4N$ operations followed by a scaling operation. In total, each element requires $4N + 1$ operations. The size of the Hessian matrix is $P \times P$. The order of calculation for the complete Hessian matrix is $\mathcal{O}\left(P^2(4N+1)\right)$. Using the QR decomposition, the rank r of a square matrix of order P can be determined in $\mathcal{O}(P^2 r)$ operations [26]. The complexity of employing the estimated Hessian rank metric is thus $\mathcal{O}\left(P^2(4N+1)\right) + \mathcal{O}(P^2 r)$. The rank estimation is done at the completion of every task, and its complexity is linear in the number of tasks μ.

3) HEURISTIC METRICS

The complexity of computing the Euclidean distance and cosine angle between the parameter vectors of two progressive tasks is $\mathcal{O}(P^2)$ [27], [28]. The complexity of

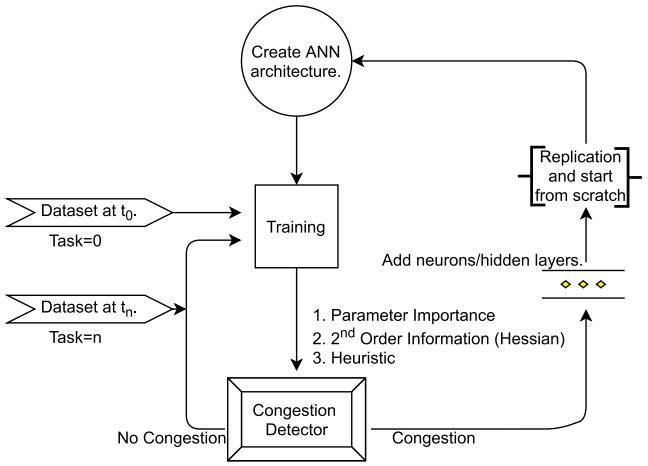


FIGURE 1. Flow chart showing the steps to build progressive model.

Jaccard's similarity is $\mathcal{O}(P \ln P)$ [29]. The complexity of linear sequence search is $\mathcal{O}(\mathcal{P})$ [30]. Note that all these complexities are linear in the number of tasks μ.

V. COMPARISON METHODOLOGY

In this section, we present the methodology, datasets, and software tools we have used to compare and contrast the various ANN congestion metrics. The results will be given in Section VI for the three progressive learning algorithms: SI, EWC, and OWM. In Section VII, the same experimental testbed will be used to evaluate a new progressive algorithm of our own, Fisher's Synaptic Intelligence (FSI), from the viewpoints of ANN congestion and catastrophic forgetting. Recall that unlike [1] where the dimension of the ANN weight vector is allowed to increase, the ANN of SI, EWC, and OWM is assumed to be fixed with the dimension of the weight vector remaining the same throughout the progressive learning.

A. EXPERIMENTAL SETUP

A flowchart showing the overall methodology for progressive learning and network congestion detection is given in Fig. 1. The setup starts with a baseline ANN architecture that is trained on an initial task $\tau = 1$ using data D_1. As a new task arrives with its own training data, the same model is incrementally re-trained using a progressive learning algorithm. The model is then sent to the congestion detector to evaluate its potential for further training. If the ANN is already congested, *congestion* = *true*, then the training sets across all the previous tasks are combined to create multiple tasks, and the ANN is expanded by adding neurons to the hidden layer or adding more hidden layers. If *congestion* = *false*, then the model remains on standby until new training data arrives. It should be noted that the expansion of the feed-forward ANN by adding a hidden layer after congestion can result in a deep neural network where the problem of vanishing and exploding gradient may occur [21]. The residual learning framework of [31] addresses this issue, as does the gated recurrent unit approach of [32]. Care weights initialization ahead of training can also mitigate gradient anomalies [33]. Such techniques are outside the scope of this paper but will be explored in our future publications.

TABLE 3. Dataset description.

Attributes	UNSW	AWID	MNIST
Description	Both normal and contemporary synthesized attack activities of LAN network traffic [34], [35].	Wireless network security dataset [36]; normal and synthesized network attack traces.	Primitive image dataset [12]; collection of gray-scale images of handwritten digits from 0 to 9.
Number of features	43	98	784
Number of labels	10	17	10
Training size	175, 341	1, 795, 575	60, 000
Validation size	82, 332	575, 643	10, 000
Feature data types	Binary, float, nominal and integer	Binary, float, nominal and integer	Integer
Some feature names	port numbers, service name, protocols, IP addresses, packets transmission statistics etc.	frame.cap_len, radiotap.present.vht, radiotap.datarate, wlan.fc.protected, wlan.qos.bit4 etc.	Pixel grayscale values.

B. DATASETS

Three datasets have been used for the experimental runs. Since our research has evolved in the context of providing AI tools for network security, we have selected two datasets from cloud network security, namely, UNSW [34], [35], AWID [36] and compared their congestion results with those of the MNIST [12] image dataset. They are briefly described in Table 3.

The original number of features in the AWID is 155, out of which 57 have raw values or invalid characters replaced with "?" across all the dataset records. The latter features are dropped from the experimental runs. To illustrate ANN progressive learning and network congestion, tasks are constituted with each task consisting of labels *randomly* selected and combined from the existing dataset. For the UNSW dataset, each task $\tau \in \{1, 2, 3, 4, 5\}$ represents a binary classification of two different security attacks, e.g., $\tau = 1$:{worms/shellcode}, $\tau = 2$:{reconnaissance/normal}, $\tau = 3$:{generic/fuzzers},and so on. Similarly, for the AWID datasets, each task performs a binary classification between two security attacks, e.g., $\tau = 1$:{normal/fragment}, $\tau = 2$:{arp/probe_request} $\tau = 3$: {chop_chop,deauthentication}, and so on. As for MNIST, the examples of binary classification tasks are $\tau = 1$:{0/1 digit}, $\tau = 2$:{2/3 digit}, $\tau = 3$:{4/5 digit}, and so on. In the experimental runs, the same task setup is used across all progressive learning models.

C. SOFTWARE TOOLS

All the experiments are performed in Python 3.5. The open-source code at [37] is used for the synaptic intelligence (SI) progressive learning model [4]. The Keras Python

TABLE 4. ANN hyper-parameters for progressive learning. The same hyper-parameter values are used across all the datasets for each of the three progressive learning algorithms: SI, EWC, and OWM.

Hyper-parameters	UNSW	AWID	MNIST
Input nodes	43	98	784
Number of dense hidden layer		1	
Number of hidden layer nodes	43	20	50
Number of output nodes	10	17	10
Batch size	64	1000	1000
Epoch per task	10	5	5
Total number of hidden-layer weights	2332	2337	39760
Activation		ReLU	
Initialization		Random	
Optimizer		Adam	
Loss function		Cross Entropy	

package [38] is used for the numerical experiments with the back-end being Tensorflow. The open-source code at [20] is used for the elastic weight consolidation (EWC) learning model [5]. Likewise, the open-source code at [39] is used for the orthogonal weight modification (OWM) learning model [6]. The Hessian matrix is computed from the outer product of gradients as given in (13). Gradients with respect to weights are calculated using *Keras callbacks*, where the finite difference of the loss function computed at the last two epochs is used. For small matrix sizes, the rank is calculated using Python's Numpy linear algebra library [40], while for large matrix sizes, the rank is calculated as the number of non-zero eigenvalues.

VI. LEARNING WITHOUT FORGETTING AND NETWORK CYBER SECURITY

In this section, the details of our comparative experiments regarding the prediction of network congestion in ANN designed to classify network security threats are given. To the best of our knowledge, this is the first such comparative study between the three progressive learning algorithms SI, EWC, and OMW, using the same computational platform with learning achieved in the networking domain, on cyber security datasets, rather than in the computer vision domain on imaging datasets. The main objective is to identify the algorithm that is capable of supporting continual learning without catastrophic forgetting so as to dynamically and autonomously detect and track emerging network cyber security threats.

A. SYNAPTIC INTELLIGENCE MODEL

The *synaptic intelligence (SI)* model is built with the specifications given in Table 4. Such specifications are obtained based on a baseline experiment using *AutoKeras* [41], [42] on the training dataset corresponding to the $\tau = 1$ task. In the progressive learning setup, only the training data corresponding to $\tau = 1$ is known, and so the ANN architecture performance is optimized for $\tau = 1$ but not for the subsequent tasks. As a result, the ANN model trained on the first task $\tau = 1$ has its highest accuracy in predicting classes. Based on (1), catastrophic forgetting is mitigated by choosing $c = 1$ [4]. The model is then trained progressively and validated after completing the training for each task. The validation is conducted not only on the completed task but also on the tasks already trained and the tasks that are still pending. The validation on the already trained tasks is meant to verify the preservation of model accuracy on prior tasks as the progressive learning proceeds. The validation on pending tasks is meant to produce reference accuracy for them against which to compare the accuracies they achieve after they are learned progressively. As an example, consider task $\tau = 3$ on the x-axis of Fig. 2 for the UNSW data. The points falling on the vertical line at $\tau = 3$ are the accuracies of all tasks that preceded $\tau = 3$ and that are still pending after $\tau = 3$. On the other hand, the accuracy plots, when read vs. the task axis τ, represent the dependence of the accuracy of a given task on the progressive learning as it proceeds from task $\tau = 1$ to $\tau = 7$. Note that the progressive task learning accuracy for the MNIST dataset is given in [4]–[6] and is not reproduced here.

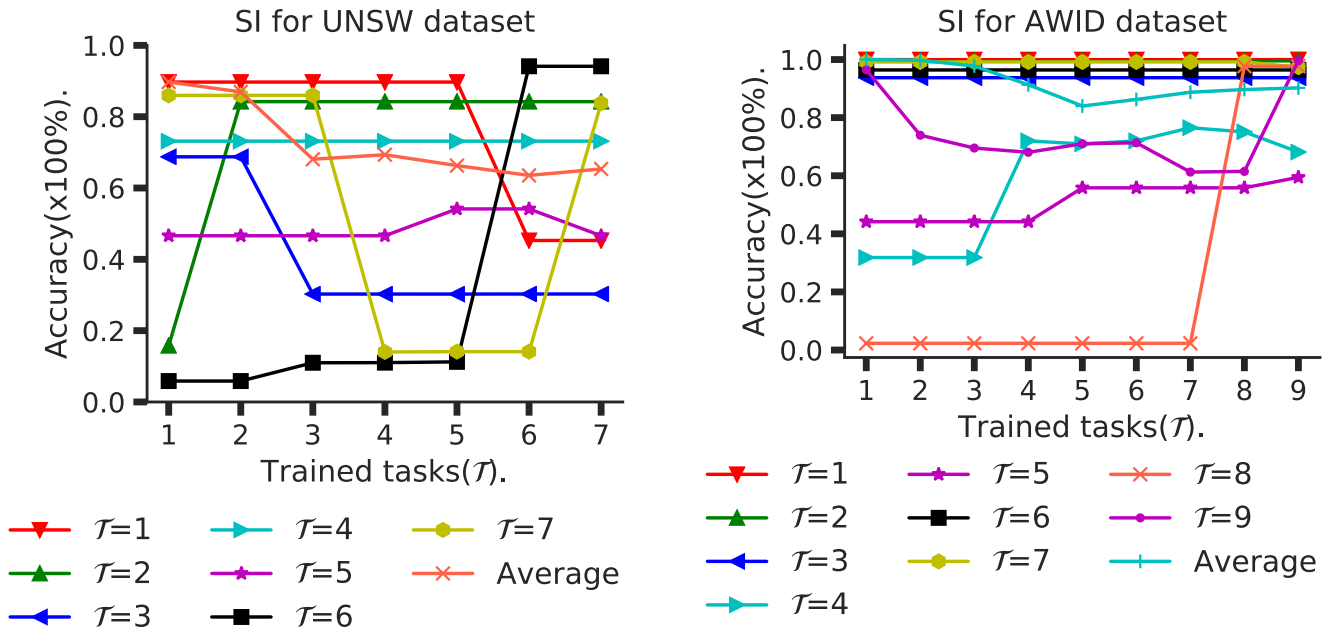
To clarify the accuracy plots of Fig. 2 further, we define the matrix $\mathbf{A} = (a_{ij})$, where the element a_{ij} is the accuracy with respect to the data of task $\tau = i$ once the task $\tau = j$ has been learned. The $\mathbf{A}$ matrix is square but not symmetric, and the arrays of Fig. 14 in the Appendix give the numerical values corresponding to the curves of Fig. 2.

One indication of catastrophic forgetting can be read from plots such as Fig. 2 by tracking a sudden drop in accuracy for a task that was previously learned adequately. This is for instance the case of task $\tau = 1$, whose accuracy drops drastically after the network learns task $\tau = 6$. Another important plot in Fig. 2 is the average accuracy computed in terms of the entries of the matrix $\mathbf{A} = (a_{ij})$ as

$$\bar{a}_k = \frac{1}{k}\sum_{i=1}^{k} a_{ik} \tag{14}$$

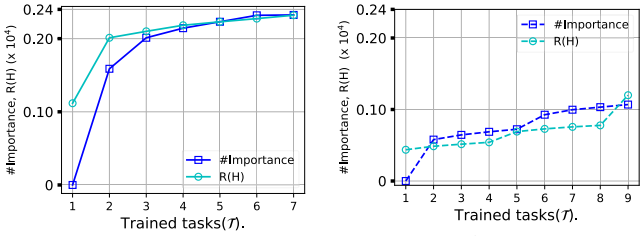
In other words, once task $\tau = k$ has been progressively learned, the average accuracy $\bar{a}_k$ computes the average of accuracies with respect to the datasets of all learned tasks up to task $\tau = k$. On the plots of Fig. 2, this average is computed as the average of the y coordinates at task k for all datasets of tasks $i \leq k$. The numerical values of these averages are shown as the last row in Fig. 14 of the Appendix where they can be computed as the average of a given column in $\mathbf{A}$ of the rows that are at or above the matrix diagonal. For instance in the AWID dataset, only a fraction of the average accuracy is lost as a result of progressive learning, as can be concluded from comparing the average accuracies of $\tau = 1$ with that of $\tau = 9$.

As for the congestion metrics described in Sections IV, they are shown in Figs. 3 and 4. It is clear from Fig. 3 that the number of important weights increases as the model learns new tasks. So does the Hessian matrix rank. For the UNSW dataset, Fig. 3a shows that the weight space has been completely occupied by the training of tasks $\tau = 1$ to $\tau = 4$. Specifically, the plot indicates that tasks $\tau = 1$ and $\tau = 2$ together have used more than half of the weight space, with



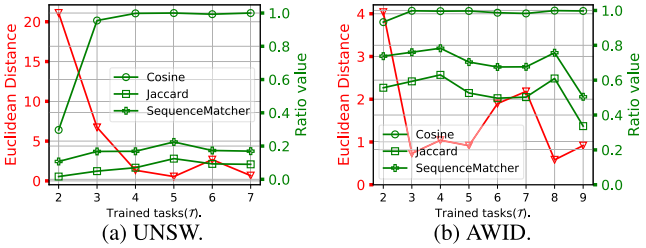
(a) UNSW (b) AWID.

FIGURE 2. Accuracy evaluation of the Synaptic Intelligence progressive learning model. Note the impact of progressive learning on maintaining accuracy across the tasks.



(a) UNSW. (b) AWID.

FIGURE 3. Importance and Hessian rank congestion metrics for the Synaptic Intelligence model of Fig. 2 for both the UNSW and AWID datasets.



(a) UNSW. (b) AWID.

FIGURE 4. Heuristic congestion metrics for the Synaptic intelligence model. Unlike Figs. 2 and 3, the tasks start at $\tau = 2$ along the x axis, while the similarity metrics are along the y axis. Note that in the numerical experiments, these metrics calculate the weight dissimilarity between two adjacent tasks τ and $\tau + 1$.

the Hessian rank exceeding 50%. Further training has caused congestion as indicated by the saturation of the Hessian rank curve. This is because most of the available directions in the 2332-dimensional weight space have been utilized, leaving only few directions for learning new tasks. As a result, tasks $\tau = \{3, 4, 5\}$ are not trainable with sufficient accuracy. Furthermore, there is a high likelihood of catastrophic forgetting in case progressive learning is pursued after $\tau = 2$ on this model. To demonstrate such likelihood, the model is trained further with tasks $\tau = \{5, 6\}$, and sure enough, catastrophic forgetting has occurred, as shown in Fig. 2a, where there is a significant degradation in the cross-validation accuracy of $\tau = \{1, 5\}$ after training for the task $\tau = 6$. Such occurrence illustrates the intuitive statement made earlier that not only does congestion negatively impact the accuracy of previously learned tasks, but also it negatively impacts the learning of future tasks.

In the AWID case of Fig. 2b, the model exhibits a large capacity to learn new tasks beyond task $\tau = 9$, as shown in Fig. 3b. This is because the model has the freedom to choose amongst several available directions for the new weights as the Hessian rank has not exceeded 50% after learning task $\tau = 9$. The heuristic metrics of Fig. 4b show that the Euclidean distance between adjacent weights decreases as the model learns new tasks. Likewise, the cosine similarity is close to 1. The qualitative heuristic metrics, Jaccard and SequenceMatcher, of the AWID dataset are higher than those of UNSW, which indicates that many more of the AWID weights are shared across the tasks than in the UNSW case. The net result of this weight sharing is a lower likelihood of network congestion.

In Table 6 and Section VIII, further comparisons will be given to contrast the congestion metrics across the progressive learning algorithms and across the datasets.

B. ELASTIC WEIGHT CONSOLIDATION MODEL

As in the SI model, the EWC model is trained progressively with several tasks. The specifications of Table 4 are again followed in the EWC model with $\beta = 2$ in Eq. (5). The congestion metrics are calculated progressively at the completion of

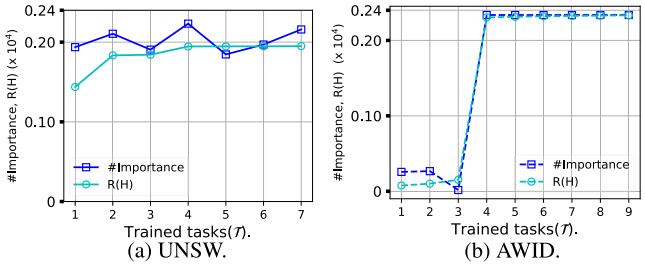


FIGURE 5. Importance and Hessian rank congestion metrics for the Elastic Weight Consolidation model for both the UNSW and AWID datasets.

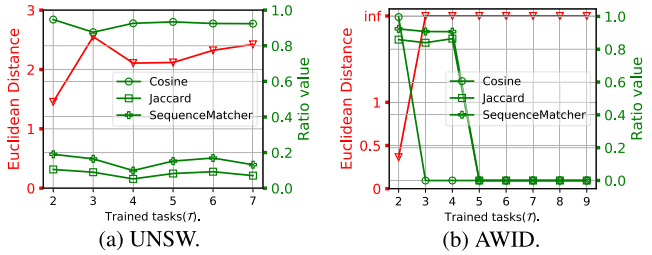


FIGURE 6. Heuristic congestion metrics for the Elastic Weight Consolidation model for both the UNSW and AWID datasets.

learning each task. In the EWC model, the Fisher information coefficient F_k^μ (7) plays a role similar to that of Ω_k^μ (2) in the SI model as a weight importance coefficient. The importance and Hessian rank congestion metrics of the EWC model are shown in Fig. 5 while its Heuristic congestion metrics are given in Fig. 6.

In Fig. 15 of the Appendix, an accuracy evaluation similar to that of Fig. 2 is given for the EWC model.

In reference to Fig. 5 and the UNSW dataset, learning task $\tau = 1$ has used more than 50% of the weights, but the network maintains some capacity to learn tasks $\tau = 2$ & 3 as well. The capacity is significantly diminished for all the subsequent tasks. In reference to Fig. 6b and the AWID dataset, the Euclidean distance metric is virtually infinite, which is consistent with a weight distribution pattern with no weight sharing beyond task $\tau = 5$.

In the EWC experimental runs, the task setup is similar to that of the SI model in that the binary classification tasks do not share any labels. However, the EWC algorithm has worse accuracy performance than SI due to statistical differences between the distribution of the Fisher's coefficients and the distribution of the SI importance parameters. These differences will be quantified in Section VII and used to motivate the introduction of a novel progressive learning algorithm using Fisher's information that achieves significant accuracy improvement with respect to EWC.

C. ORTHOGONAL WEIGHT MODIFICATION MODEL

As in the SI and EWC model, the OWM model is trained progressively with the specifications of Table 4. All the congestion metrics are calculated after training each task. The OWM model has no importance coefficients like Ω_k^μ for SI or F_k^μ for EWC. Nor does it use a surrogate loss $L_{\tau\leq\mu}$ as in SI and EWC. The congestion metrics of the OWM model are shown in Fig. 7.

Furthermore, an OWM accuracy plot similar to that of Fig. 2 has been generated and included as Fig. 16 in the Appendix. The OWM accuracy will be compared to that of SI and EWC in the next section.

VII. FISHER SYNAPTIC INTELLIGENCE (FSI)

In this section, we introduce an improved version of EWC called Fisher Synaptic Intelligence (FSI). The main novelty of FSI is to replace the original penalty terms of EWC based on the raw Fisher's information coefficients with summations that are similar to the ones used to compute the importance coefficients of SI. Numerical experiments show that FSI results in significantly improved accuracies with respect to EWC using the same set of tasks as the SI algorithm. This section also contains a comparison among the various progressive learning algorithms: SI, EWC, OWM, and FSI, from the viewpoint of their accuracies.

A. MOTIVATION

As mentioned in Section VI-B, when both SI and EWC are compared on the same set of cyber security tasks, EWC suffers a loss of accuracy. To gain more insight into this loss, the histograms of the SI penalty coefficients (importance parameters) are generated and plotted in Fig. 8 for the UNSW dataset. This histogram has a long tail, indicating that some of the Ω_k^μ penalty parameters are large. In Fig. 8a, the Ω_k^μ histogram of SI is compared with the F_k^μ histogram of EWC to show that the latter is much narrower than the former. The large penalty parameters on the tail of the SI histogram force the optimization algorithm to keep some of the weights close to their previous values while using the weights of the small parameters to learn new tasks. This is not the case for EWC where the majority of the penalty coefficients (Fisher's information) are small and, in fact, concentrated close to zero. The EWC optimization algorithm treats each new task as essentially as a learning-from-scratch case, thus resulting in the EWC accuracy loss.

B. IMPROVING EWC

To address the EWC loss of accuracy, we replace the raw Fisher's coefficient at task μ with the cumulative sum of all the Fisher's coefficients up until task μ according to the formula

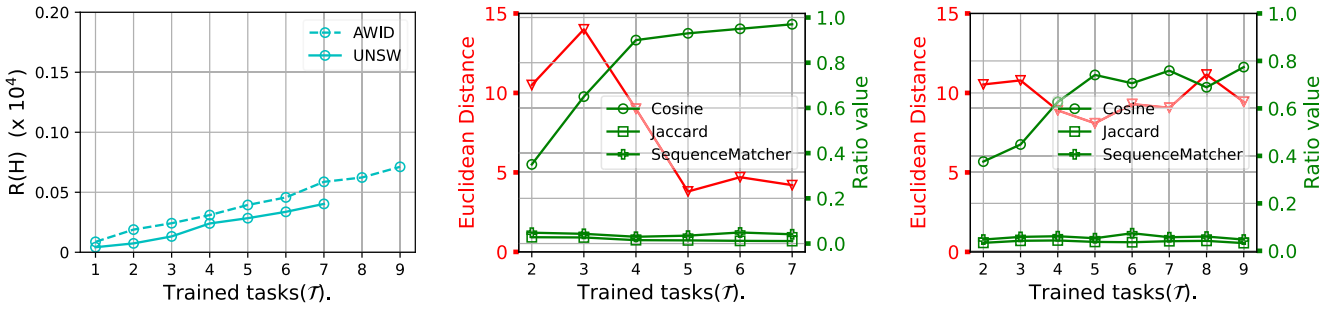
$$\Phi_k^\mu = \sum_{\tau\leq\mu} F_k^\tau \tag{15}$$

with the objective function becoming

$$\tilde{L}_{\mu+1}(\boldsymbol{\theta}) = L_{\mu+1}(\boldsymbol{\theta}) + c\sum_{k=1}^{P} \Phi_k^\mu(\theta_k^\mu - \theta_k)^2 \tag{16}$$

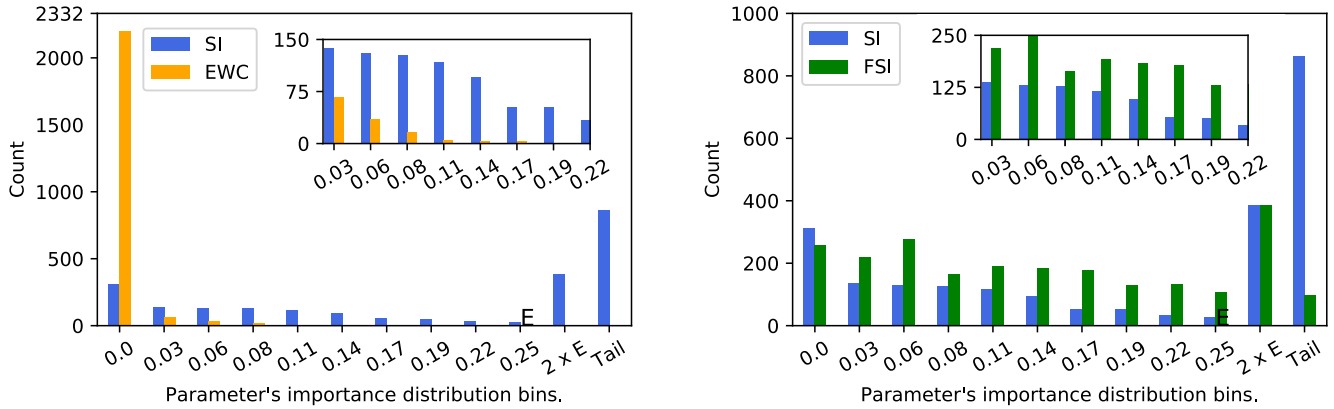
Note that the summation (15) is similar to that of (2) of SI, and as a result, the objective function of (16) along with (15) is called Fisher Synaptic Intelligence (FSI). The computation of Φ_k^μ can be done recursively according to

$$\Phi_k^{\mu+1} = \Phi_k^\mu + F_k^{\mu+1} \tag{17}$$



(a) Hessian Rank (b) UNSW. (c) AWID.

FIGURE 7. **Congestion metrics of the OWM model for both the UNSW and AWID datasets.**



(a) Comparison of SI's Ω_k^{μ} and EWC's F_k^{μ} histograms on the UNSW dataset at $\tau = 5$.

(b) Comparison of SI's Ω_k^{μ} and FSI's Φ_k^{μ} histograms on the UNSW dataset at $\tau = 5$.

FIGURE 8. **Comparisons of the penalty coefficients for SI, EWC, and FSI for the UNSW dataset at task $\tau = 5$. The 'E' marker denotes maximum value of the EWC distribution. The '2×E' marker is twice the value of 'E'. All the values larger than 2×E are included in a 'Tail' bin. The subplot inside each plot is a zoom-in on the distribution details. Please note the scale difference of the y-axis in the two diagrams.**

with no need to store the training data corresponding to tasks prior to μ. The net effect of using Φ_k^{μ} instead of F_k^{μ} is to broaden the histogram of the penalty coefficients of the objective function. This is illustrated in Fig. 8b, where the FSI histogram is shown to be close to the SI one and therefore much broader than the EWC histogram, thus increasing the penalty on some of the terms of the proxy loss function. As in SI, this increase will result in a differentiated update of the weights with the new tasks triggering changes in the weights having small penalty coefficients. More information on the SI, EWC, and FSI histograms using the UNSW dataset is given in Figs. 18b and 19 of the Appendix.

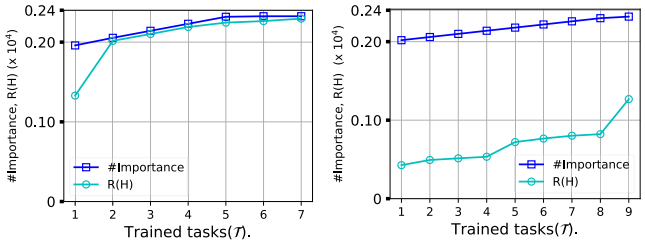
The FSI model has been trained with tasks identical to those of the SI and EWC models of Sections VI-A and VI-B. The FSI accuracy plot is given in Fig. 17 of the Appendix. The FSI congestion metrics are shown in Figs. 9 and 10.

For UNSW, the behavior of FSI is similar to that of the SI model of Fig. 2a, where tasks $\tau = 3$ and 4 have not been able to train due to congestion and task $\tau = 1$ encounters catastrophic forgetting (drop in accuracy at $\tau = 6$). The $n(F)$ and Hessian rank plots show full weight space utilization around $\tau = 3$. The Euclidean distance plot of Fig. 10 shows monotonous decrease down to $\tau = 5$, after which the distance increases, signaling congestion. The Jaccard and SequenceMatcher show minimal similarity ratio between weights of adjacent with a further decrease after $\tau = 5$. The AWID model is not as congested as the UNSW one. In particular, the Hessian rank indicates a significant capacity to learn new tasks, yet the number of non-zero penalty coefficients shows near-full weight utilization starting at $\tau = 6$. This situation is explained by the presence of weight sharing among the tasks. In this particular case, the non-zero penalty coefficients give a false congestion alarm, while the Hessian metric is a much better indicator of progressive learning capacity. As for FSI accuracy, more will be said about it in the next paragraph.

C. SI, EWC, OWM, AND FSI COMPARISONS

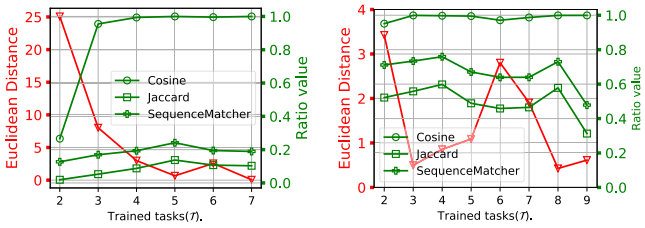
In this paragraph, the accuracy comparisons of the three prior models on progressive learning: SI, EWC, and OWM, are given along with the accuracy results of the proposed FSI model. Only average accuracies as given in (14) are considered in the comparisons. Aside from the new FSI progressive learning model, one major contribution of this paper is that all comparisons are conducted on the same platform using the same datasets and the same task definition for all the algorithms. The accuracy comparison results are summarized in Fig. 11. We note the following:

1) The EWC model is less accurate than the SI model with a significant drop beyond task $\tau = 3$.



(a) UNSW. (b) AWID.

FIGURE 9. Fisher synaptic intelligence (FSI) model evaluation with rigorous congestion metrics for the UNSW and AWID datasets.



(a) UNSW. (b) AWID.

FIGURE 10. Fisher synaptic intelligence (FSI) model evaluation with heuristic congestion metrics for the UNSW and AWID datasets.

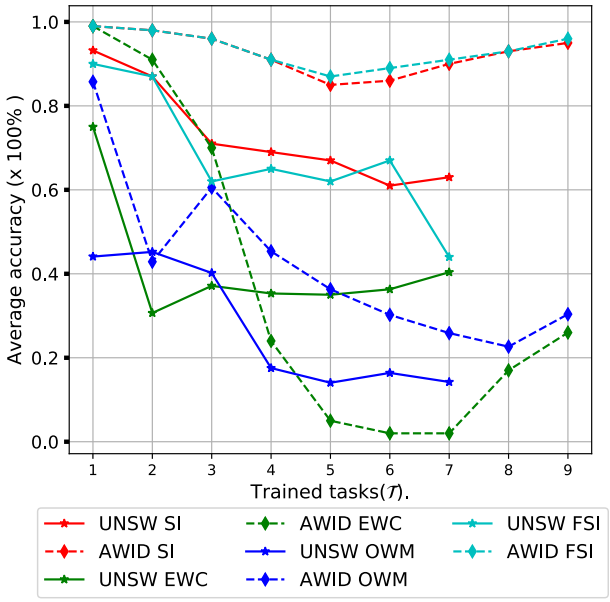


FIGURE 11. Comparison of average accuracies for the SI, EWC, OWM, and FSI progressive learning algorithms. Refer to Section VI-A and Figs. 2 and 14 for the interpretation of this plot.

2) The OWM model is also less accurate than the SI model and more prone to early congestion. The OWM accuracy drops after task $\tau = 4$ for both datasets UNSW and AWID.
3) The use of FSI results in a significant improvement of accuracy, especially for the AWID dataset.

It is important to note that the above results are dataset-dependent. When the MNIST dataset is used, [6] reports an improvement in accuracy of OWM over both SI and EWC. On the other hand, such an improvement has not been obtained with UNSW and AWID. To gain more insight into this dependence on the dataset, we have listed, in Table 5, the matrix rank of $\mathbf{O}_\mu(0)$, the sub-matrix of the projection matrix $\mathbf{O}_\mu$ (see (10) that is conformal to the weights of the input layer $l = 0$ of the ANN. For MNIST, this rank decreases with progressive learning, but for UNSW and AWID, it remains near full rank. This suggests that the MNIST data is correlated task-to-task, resulting in the projection space becoming of lower dimension as the learning proceeds. However, the data of UNSW and AWID are independent task-to-task, resulting in the projection space corresponding to the weights of the input layer maintaining nearly the same dimension.

In reference to Figs. 3 and 9, the penalty coefficients increase monotonically with task learning for the SI and FSI models, but for EWC, their behavior is mixed, as shown in Fig. 5. OWM has no such coefficients.

TABLE 5. Orthogonal projection matrix across tasks for OWM algorithm. Size of orthogonal matrix for UNSW is $\langle O_\tau(0)\rangle = [43 \times 43]$, AWID is $\langle O_\tau(0)\rangle = [98 \times 98]$, MNIST is $\langle O_\tau(0)\rangle = [784 \times 784]$.

Task τ	$O_\tau(0)$ matrix rank		
	UNSW	AWID	MNIST
1	43	98	769
2	43	98	749
3	42	97	734
4	42	97	724
5	42	97	713
6	41	97	-
7	41	96	-
8	-	96	-
9	-	96	-

VIII. DESIGN AND DOMAIN-SPECIFIC ISSUES

A. WEIGHT SHARING AND LABEL GROUPING

Once the ANN model undergoes congestion, the inference engine has to be discarded, and a new engine should be designed with additional hidden layers, as shown in Fig. 1. In order to avoid early congestion in the re-designed inference engine, the labels should be packed together to improve weight space usage. Instead of randomly selecting such labels, a label packing algorithm is more likely to reduce over-fitting and improve the overall generalization accuracy. Such label packing can be derived based on similarity metrics applied to the ANN weights of a given pair of tasks. An approximate measure of similarity between ANN weights can maximize the number of labels that can be packed into an ANN model without risking congestion or catastrophic forgetting.

To evaluate label packing, we have used the UNSW dataset. It has 43 feature columns and an *'attack_category'* column [34] containing 10 classes that are used as labels. One-hot encoding [43] has been applied to the *'attack_category'* column, which has resulted in a 10-bit binary code for each label and 10 binary classification models. Each model has one output node containing the sigmoid activation function. The other hyper-parameters are as given in Table 4. A standard ANN learning model is trained for each binary classifier, and model weights are collected. The binary classifiers are paired, and the four similarity metrics,

explained in Section IV-D, are applied to the weights of each pair, thus resulting in 10 matrix for each similarity metric. These matrices are visualized in Figs. 12a and 12b for the Euclidean and Cosine metrics, respectively. These two metrics are able to detect the pairwise weight similarity. On the other hand, the Jaccard and SequenceMatcher fail to detect such similarity. Illustrations similar to those of Fig. 12 for Jaccard and SequenceMatcher have been included in Fig. 20 of the Appendix.

Based on Figs. 12a and 12b, the weights corresponding to *Worm* and *Analysis* pair of labels have the highest similarity. So do the weights corresponding to the *DoS* and *Backdoor* pair. Therefore each pair of labels can be packed together in a single binary classification task with the expectation that some of their ANN weights will be shared. The labels are paired as follows: $\tau = 1$:{'Worm'/'Analysis'}, $\tau = 2$:{'DoS'/'Backdoor'}, $\tau = 3$:{'Normal'/'Exploits'}, $\tau = 4$:{'Shellcode'/'Generic'}, $\tau = 5$:{'Reconnaissance'/'Fuzzers'}. A progressive learning model is created using these paired labels with a set of 5 binary classification tasks. Such tasks are fed sequentially to the SI learning model with the specifications given in Table 4. Another SI learning model with 5 binary classification tasks is built with the same specifications but with labels paired randomly as in Section VI-A and Fig. 2a. As illustrated in Fig. 13, similarity-based label packing results in better accuracy than random packing of labels. Accuracy comparison is shown in Fig. 13(1), while two congestion metrics are shown in Fig. 13(2&3). Finally, the differences in the Hessian rank and penalty coefficient metrics are shown in Fig. 13(4). The similarity-based label packing has distinct advantages in comparison with random label packing, including better accuracy and lower congestion risk. Because of the latter advantage, similarity-based label packing may be able to accommodate more tasks for a given ANN architecture.

While label packing results have been shown for the UNSW dataset for SI model, similar results carry over to the other datasets and to the EWC, OWM, and FSI progressive learning models.

B. NETWORK SECURITY VS. IMAGE RECOGNITION

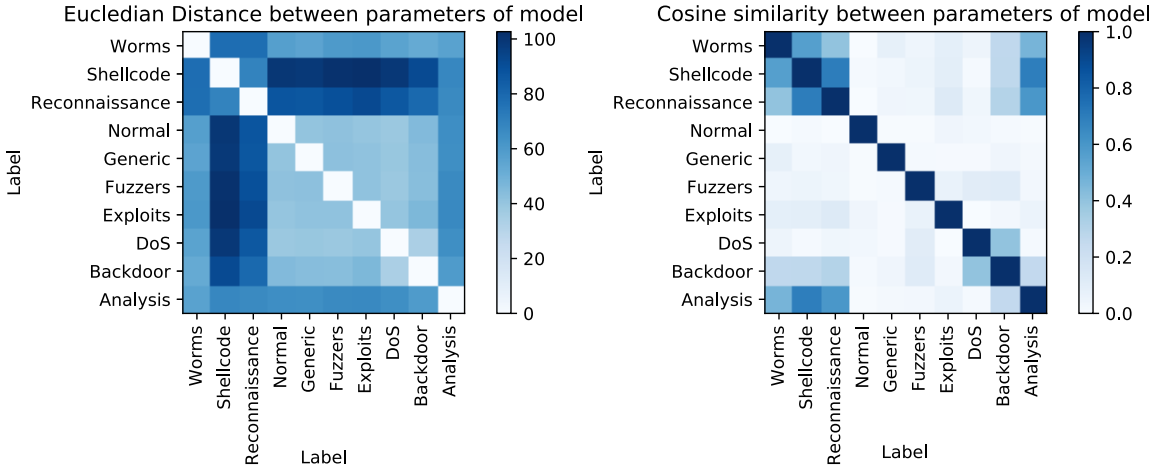
Published articles on Progressive Learning have often used image datasets to illustrate its main features and advantages. In particular, the MNIST dataset has been the most used, followed by the CIFAR and Imagenet datasets. On the other hand, publications on Progressive Learning using cloud network security datasets are few and far between [44].

ANN machine learning has shown distinct advantages over ''expert-system'' learning for the detection of network malware and intrusion. Such neural network models have to be maintained and upgraded post-deployment in order to adapt to newly encountered attacks. Progressive learning provides a rigorous framework for enabling such adaptation in near real time and with reasonable computational complexity. ANN models for cloud security datasets are typically much smaller than their image-recognition counterparts. As an example, the UNSW and AWID cloud cyber security datasets have 43 and 98 features, respectively, while a high-definition image may have up to 3, 145, 728 pixels, each being a feature, with the input ANN layer having one input node per pixel. As a result, image classification models have a larger number of weights than cyber-attack classification ones, with the consequence that progressive learning is easier to implement in the cyber security case.

While results related to MNIST progressive learning have been extensively reported [4]–[6], [15]–[18], we have used our platform to perform our own MNIST experiments so as to facilitate apple-to-apple comparisons with the results obtained for the UNSW and AWID cyber security datasets. Our results are summarized in Table 6. Note that the MNIST tasks used in the generation of Table 6 are binary classification tasks similar to those defined for the UNSW and AWID datasets. In other words, they are not the *identification* tasks used in [5] which are meant to address a different classification question, namely, ''Does any of the digits between 0 and 9 show up in this image?'' When symmetry transformations are applied to an MNIST image, the digit is still present but the underlying pixel distribution is changed. On the other hand, the sequence of 5 non-overlapping binary tasks, $\tau = \{0, 1\}, \{2, 3\}, \{4, 5\}, \{6, 7\}, \{8, 9\}$, of Table 4 addresses the binary classification question ''Is the digit present in the image *i* or *j*?''.

It is conceivable to construct a cyber security detection model to check if a given incoming attack is one of a given set of learned attacks. However, feature permutation within each attack in the training set is not appropriate due to the non-homogeneity of features across various attacks. Furthermore, each feature in the attack data has a physical interpretation of its own, and two different attacks may be characterized with sets of non-overlapping features. The issue of task definition has a far-reaching impact on the outcome of progressive learning and catastrophic forgetting. As an example, the behavior of the EWC model on tasks, each made of a collection of transformed MNIST images [5] is quite at variance with its behavior on tasks, each made of a binary classification of pair of MNIST images. The FSI improvement introduced in Section VII is meant to address this difference. Indeed, as shown in Table 6, the Hessian rank of the FSI model is much lower than that of the EWC one, which is more in line with the expectation that the CNN model of MNIST is still far away from catastrophic forgetting. When it comes to progressive learning, the differences between the image domain and the cyber security domain may be summarized as follows:

1) **Task Definition:** As mentioned above, there are several methods for defining progressive learning tasks, and the specific method used depends on the precise statement of the classification problem the neural network is supposed to solve. It also depends on whether the data features are homogeneous or heterogeneous. By and large, in the imaging domain, the feature space is homogeneous, which allows flexibility in the selection



(a) Euclidean distance between model weights.

(b) Cosine similarity between model weights.

FIGURE 12. Quantitative similarity between model weights.

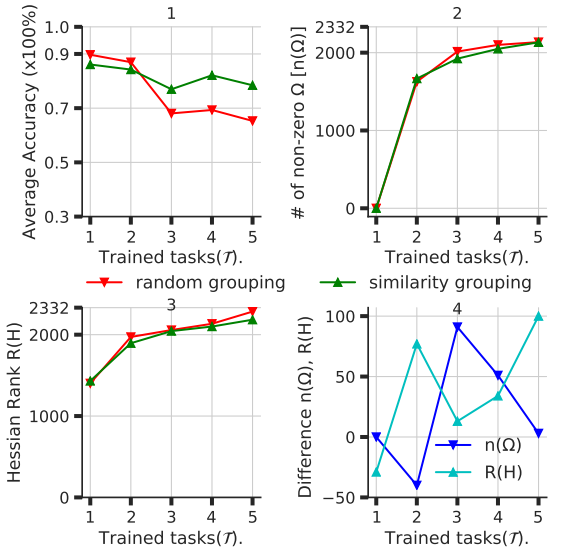


FIGURE 13. Progressive learning for the UNSW dataset where each task contains labels that are grouped using Euclidean and cosine similarity. The label-packing model is compared with the one built in Section VI-A, where labels were selected randomly for each task. Plot 1 compares the average accuracies of the two models. Plots 2 and 3 compare their congestion metrics. In Plot 4, the dark-blue curve represents the difference in SI penalty coefficients, while the light-blue curve represents the difference in Hessian ranks.

of tasks. In the cyber security domain, the feature space is essentially attack-dependent, which makes the options for task selection quite restricted.

2) **Network Architecture:** Convolutional neural network (CNN) models are heavily used for image classification [13], where the image pixels are manipulated through filtering, cropping, sub-sampling, and pooling layers. The weights of each filter are tuned in the back-propagation learning phase. A large number of layers increases the number of weights but also produces highly accurate progressive learning models. For instance, an accuracy of more than 95% has been achieved with CIFAR [6]. On the other hand, CNN is not used for cyber security datasets. The ANN in the latter domain is rather shallow but fully connected with the weight space having a lower dimension than that of CNN.

3) **Congestion Metrics:** The size of the Hessian matrix is $[P \times P]$, where P is the total number of ANN parameters. In CNN image recognition, the value of P increases with the number of layers, the number of filters, and the number of weights in each filter, which ultimately makes the size of the Hessian matrix excessively large for even approximate rank computation. For instance, the CIFAR model contains two *convolutional*, two *max-pooling*, and a densely connected output layer, totaling 1, 276, 508 weights [37], with the resulting Hessian matrix rank calculation becoming prohibitive. Heuristic congestion metrics are therefore more convenient for imaging applications. On the other hand, in cyber security applications, the Hessian rank, along with the other heuristic congestion metrics of Section IV, is applicable with very reasonable computational cost.

In light of the above differences, the cyber security domain is quite amenable to the usage of progressive learning models that are computationally efficient and highly accurate. Such models have the additional advantage that the onset of catastrophic forgetting is highly predictable due to the use of the loss function Hessian rank in evaluating network congestion.

C. RELATED WORK

Machine learning frameworks for cyber security threat identification have received significant attention over the last decade. One of the most recent studies [45] uses several machine learning methods to develop an intrusion detection system for the IoT domain. The framework consists of standard one-shot learning organized hierarchically with the first level using deep autoencoders to *detect* traffic with malicious activity and the second level using random forests, Naive Bayes, and multi-layer perceptrons to *classify* anomalous traffic into several categories of cyber security threats. The

TABLE 6. Comparison of congestion metrics using SI [4], EWC [5], our proposed FSI, and OWM [6]. Rows corresponding to "Penalty Coefficients" and "Hessian Rank" are measured as percentages with respect to their total values after training on the last task. Example: Hessian Rank = $\frac{R(H)}{P}$ × 100% where P is the number of weights, which is the maximum number the rank can reach. Note that not all heuristic metrics have an upper bound. For instance, the Euclidean distance may be infinite. Using heuristic metrics, congestion is quantified by measuring metric deviation between the first and last trained tasks.

Congestion Metric	Algorithm	UNSW	AWID	MNIST
Penalty Coefficients (%)	SI	99.39	62.43	90.16
	EWC	92.58	99.98	53.10
	FSI	98.88	97.69	99.99
	OWM	-	-	-
Hessian Rank (%)	SI	97.85	56.56	96.53
	EWC	83.40	99.45	99.99
	FSI	98.11	55.41	75.62
	OWM	15.99	20.92	30.82
Euclidean	SI	19.853	4.756	8.932
	EWC	4.327	∞	$1.6E32$
	FSI	26.465	5.549	6.247
	OWM	25.332	10.63	45.028
Cosine	SI	0.336	0.982	0.772
	EWC	0.726	∞	0.01
	FSI	0.267	0.976	0.869
	OWM	0.217	0.571	0.148
Jaccard	SI	0.009	0.246	0.052
	EWC	0.014	0.0	0.058
	FSI	0.0096	0.259	0.052
	OWM	0.003	0.032	0.0005
Sequence-Matcher	SI	0.003	0.381	0.097
	EWC	0.028	0.0	0.112
	FSI	0.0068	0.401	0.097
	OWM	0.005	0.047	0.0

reader is referred to [45] and its reference list for more pointers to the prior art on using machine learning for cyber security threat detection. In this section, we focus on the use of multi-task progressive learning for incremental threat detection and identification and the risks of catastrophic forgetting that ANN's may incur in the cyber security context.

As mentioned in Section IV, the Hessian matrix of the loss function can be used as the basis for predicting catastrophic forgetting. The closest work to this Hessian matrix approach is [46], where a technique called *curvature propagation* (CP) has been proposed to efficiently calculate Hessian approximations. CP uses a rank-one approximation of the Hessian matrix based on the gradient vector. Higher-order approximations are dependent on the number of CP iterations. In contrast, this paper provides higher-rank approximations in a computationally efficient manner because it uses the ANN weight gradients which are pre-calculated during progressive learning. In [47], the exact Hessian matrix is calculated using multiple forward and backward propagations through the network. The number of operations per propagation is quadratic in the number of weights, which is also the case for accurate Hessian calculation. An alternative method for approximating the Hessian matrix is given in [48], where the Levenberg-Marquardt (LM) optimization algorithm is used to provide a Gauss-Newton approximation of the Hessian. LM is used to locate the optimal weights under an L_2 regularization constraint. Note that the Gauss-Newton approximation requires the computation of the Jacobian of the cost function, and as such, is similar to the CP approximation.

In relation to weight sharing amongst tasks (see Subsection VIII-A), there are two methods of multi-task learning according to whether weight sharing is hard or soft, as explained in [49]. Hard sharing amounts to a complete sharing of the hidden layers amongst all tasks while keeping the output layers task-specific. The multi-task learning framework DISTILLER [50] is an example of hard sharing. To learn any task, it employs a two-phase learning procedure comprising a pre-training phase and a fine-tuning phase. The pre-training phase *distills* discriminative information from each task to generalize the network traffic dataset representation. The fine-tuning phase *adjusts* the parameters of task-specific layers while freezing the parameters of the shared layer. On the other hand, in soft sharing, one model with its own weights is dedicated to each task. The distances between model weight vectors are then regularized to come up with weight vectors that are as close as possible [1]. The progressive learning algorithms of this paper uses soft weight sharing. A process of identifying the relationship between tasks and its impact on multi-task learning has been presented in [51], where regularization and soft weight sharing are used to help predict task synergies and facilitate their grouping for multi-task learning. Another example is [1], which also uses a combination of regularization and soft weight sharing. The learning of a new task is accomplished in three phases: a growth phase, a warm-up phase, and a full optimization phase. In the growth phase, the output layer is augmented with additional nodes and, therefore, additional weights to accommodate the new task. The warm-up phase initializes the new weights in the output layer while freezing all other weights. The full optimization phase tune all weights jointly using a regularization penalty that implicitly enforces soft weight sharing. The impact of training secondary tasks in parallel with primary tasks on the convergence and accuracy of the latter has been analyzed in [52]. The analysis has been performed for various applications, including topic prediction, sentiment analysis, and hash-tag recommendation. Two methods have been used for parallel training. The first is based on the sharing of a long-short-term memory (LSTM) model between the primary and secondary tasks. The second method is based on having the secondary task control the training of an output LSTM model of the primary task. In transfer learning [53], [54], the training that a given model has already received from prior tasks is used to guide its training on new tasks.

IX. CONCLUSIONS AND FUTURE WORK

In this paper, a unified learning-without-forgetting framework for network cyber security applications has been investigated using four different progressive learning algorithms. Three of these algorithms, namely, Synaptic Intelligence (SI), Elastic Weight Consolidation (EWC), and Orthogonal Weight Modification (OWM), are from the prior art, but in the proposed framework, they have been applied to the network cyber security domain for the very first time. The fourth algorithm, Fisher Synaptic Intelligence (FSI), is a novel algorithm

Validation

Progressive learning

Tasks	1	2	3	4	5	6	7
1	0.879	0.897	0.897	0.897	0.897	0.452	0.452
2	0.157	0.819	0.842	0.842	0.842	0.842	0.842
3	0.687	0.687	0.241	0.238	0.238	0.303	0.303
4	0.731	0.731	0.731	0.731	0.731	0.731	0.634
5	0.466	0.466	0.466	0.466	0.54	0.54	0.466
6	0.058	0.058	0.059	0.059	0.059	0.941	0.941
7	0.859	0.859	0.859	0.141	0.143	0.143	0.859
Avg.	0.897	0.858	0.66	0.677	0.649	0.635	0.642

(a) Accuracy matrix for UNSW.

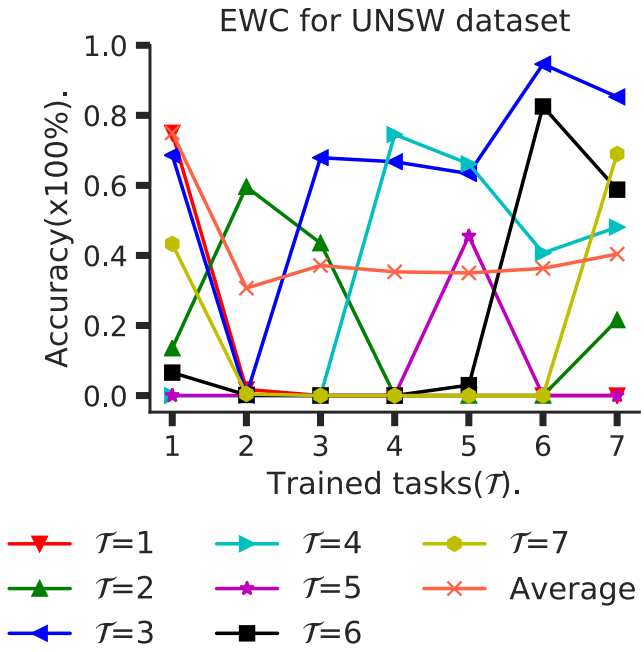
Validation

Progressive learning

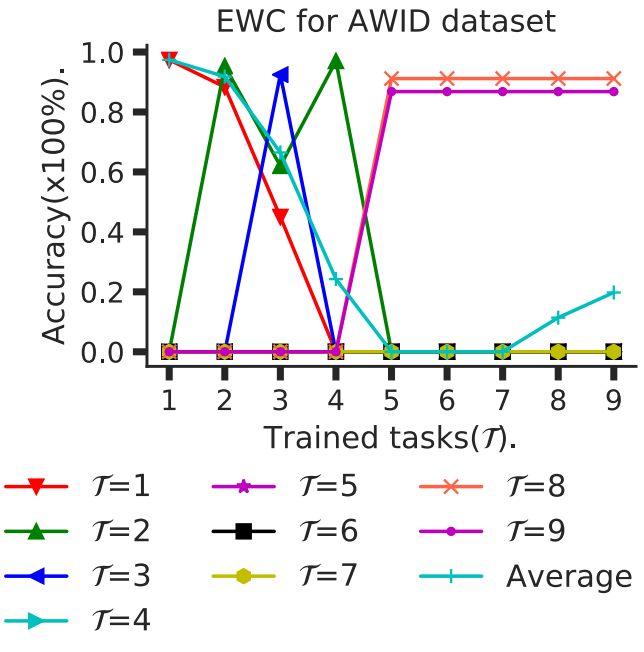
Tasks	1	2	3	4	5	6	7	8	9
1	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999
2	0.994	0.994	0.994	0.994	0.994	0.994	0.994	0.994	0.994
3	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93
4	0.318	0.318	0.318	0.746	0.719	0.721	0.755	0.748	0.681
5	0.431	0.431	0.431	0.431	0.572	0.572	0.568	0.568	0.582
6	0.963	0.963	0.963	0.963	0.963	0.963	0.963	0.963	0.963
7	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.989
8	0.022	0.022	0.022	0.022	0.022	0.022	0.022	0.977	0.977
9	0.977	0.7836	0.723	0.701	0.728	0.735	0.675	0.678	0.998
Avg.	0.999	0.996	0.974	0.917	0.842	0.863	0.8868	0.896	0.901

(b) Accuracy matrix for AWID.

FIGURE 14. Accuracy matrices corresponding to the plots of Fig. 2. Note that the font colors correspond to the color legend of the plots.



(a) UNSW

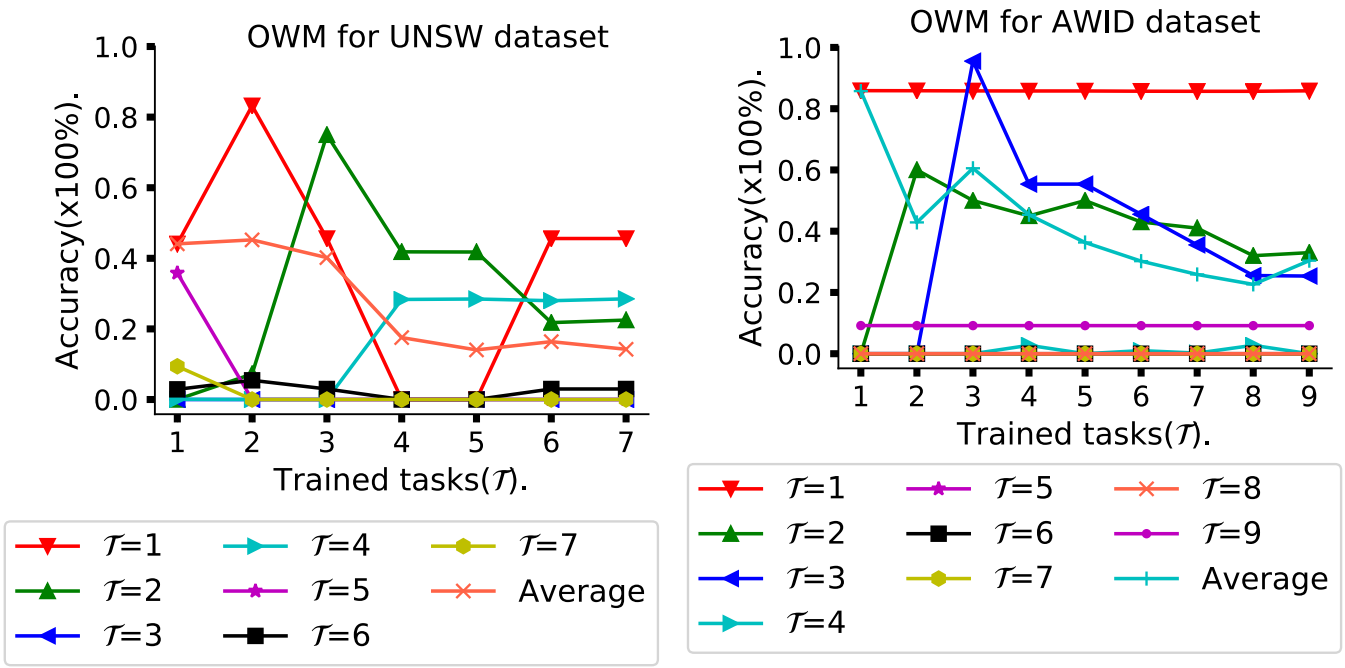


(b) AWID

FIGURE 15. Accuracy evaluation of the Elastic Weight Consolidation (EWC) progressive learning model.

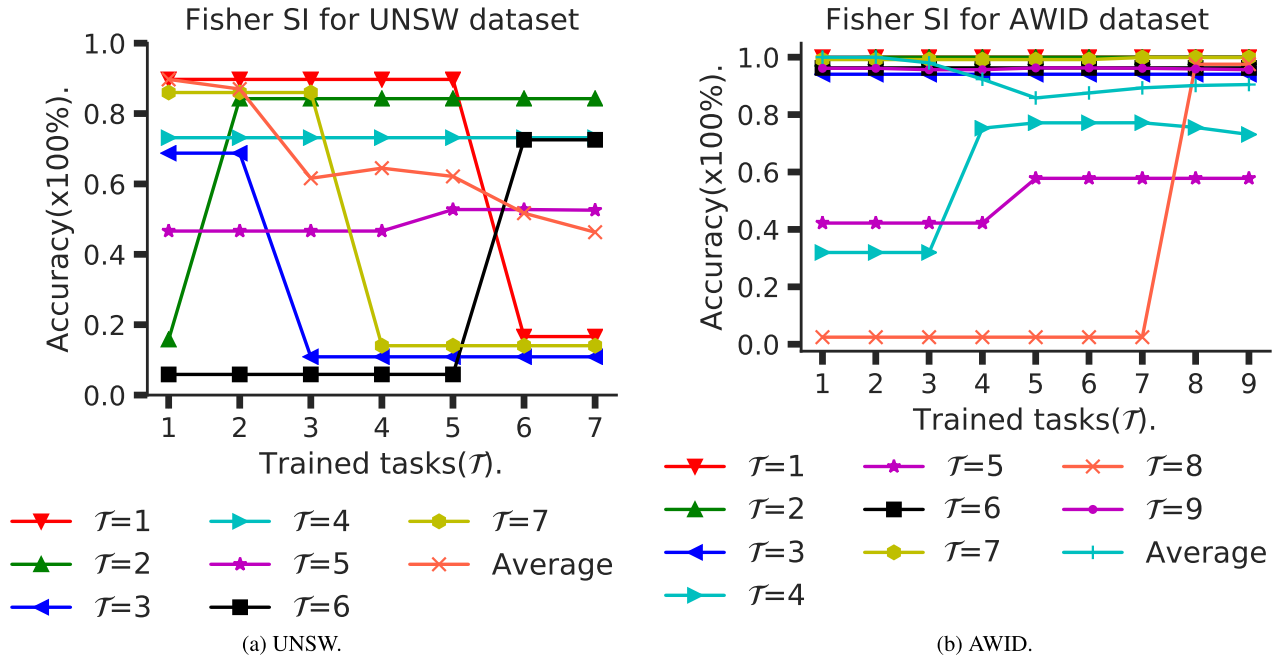
that is based on a statistical analysis of the coefficients of the proxy loss functions of SI and EWC. All these algorithms have been compared and contrasted from the viewpoints of accuracy and progressive-learning capacity using datasets from the network cyber security domain. In particular, three methods based on penalty coefficients, approximate Hessian rank, and weight similarity metrics have been investigated for detecting the onset of congestion and catastrophic forgetting in all four algorithms. The results have shown adequate alignment of these three criteria in detecting these phenomena in cyber security datasets. The paper has further provided new insights into the differences between SI and EWC progressive learning performance based on the statistical analysis of their proxy loss function coefficients. This analysis was the foundation of the novel FSI algorithm, which uses Fisher's information to compute the proxy loss coefficients but yields a progressive learning performance that is comparable to SI on cyber security datasets. An additional aspect of this paper was the use of similarity measures to conjure up label packing techniques that were shown to result in better prediction accuracy than random label packing.

This work can evolve in several interesting directions. First, the proposed framework is based on the premise that the ANN architectures are fixed and that the various progressive-learning tasks have to be accommodated within a parameter space of fixed dimension. The catastrophic forgetting criteria proposed in this paper make fundamental use of this fixed-architecture assumption. An interesting future direction is to explore such criteria for learning-without-forgetting frameworks that include architectural growth under the AutoML paradigm of [1]–[3]. Second, the training of all the tasks in the proposed framework has assumed that the hyper-parameters of each algorithm are fixed throughout the progressive learning process. This assumption may not be optimal as task-to-task variability might require task-based hyper-parameter tuning to achieve faster convergence or better accuracy. It is, therefore, important that future work explores the incorporation of hyper-parameter tuning paradigms [55], [56] within the context of progressive learning and investigates its impact on the learning-without-forgetting framework for cyber security applications. Third, in time-critical applications, such



(a) UNSW. (b) AWID.

FIGURE 16. Accuracy evaluation of the Orthogonal Weight Modification (OWM) progressive learning model.



(a) UNSW. (b) AWID.

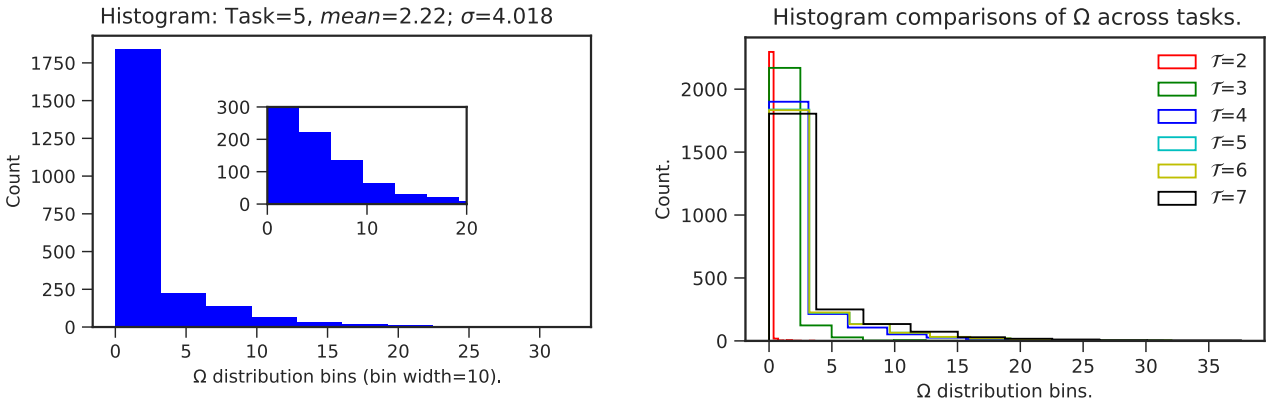
FIGURE 17. Accuracy evaluation of the Fisher Synaptic Intelligence (FSI) progressive learning model.

as identification and classification of cyber network threats, the computational efficiency of the machine-learning algorithms is of paramount importance. Hardware acceleration of machine learning has been a major research area over the past few years, especially in the context of imaging and computer vision [57], [58]. However, to the best of our knowledge, very little work has been reported on the hardware acceleration of progressive learning algorithms and even less on their accelerations in the context of cyber security applications.

APPENDIX

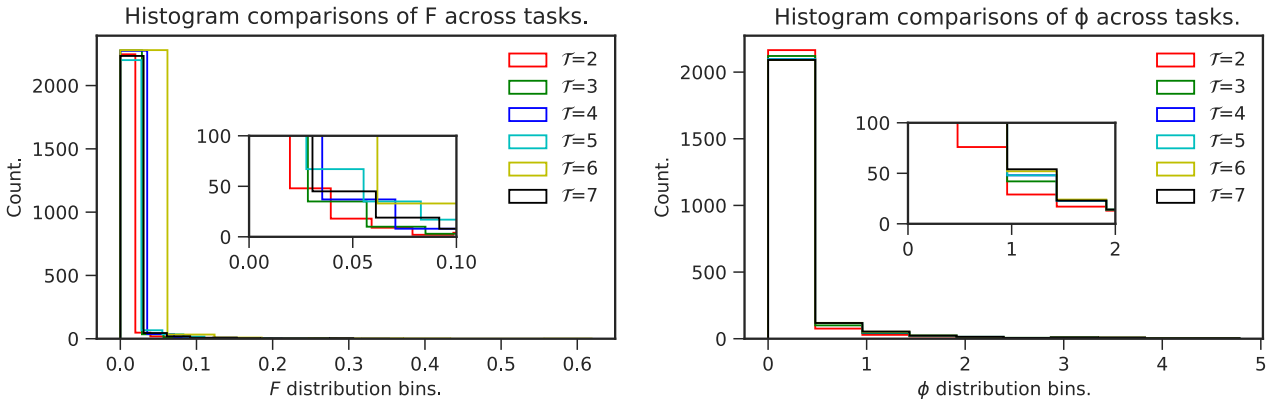
This appendix contains additional figures that are meant to provide further support to the various claims made in the body of our paper:

1) Fig. 14 displays the array of numerical values corresponding to the Synaptic Intelligence (SI) model curves of Fig. 2.
2) Fig. 15 shows accuracy evaluations of the Elastic Weight Consolidation (EWC) model using the UNSW and AWID datasets. This figure is a counterpart of Fig. 2.



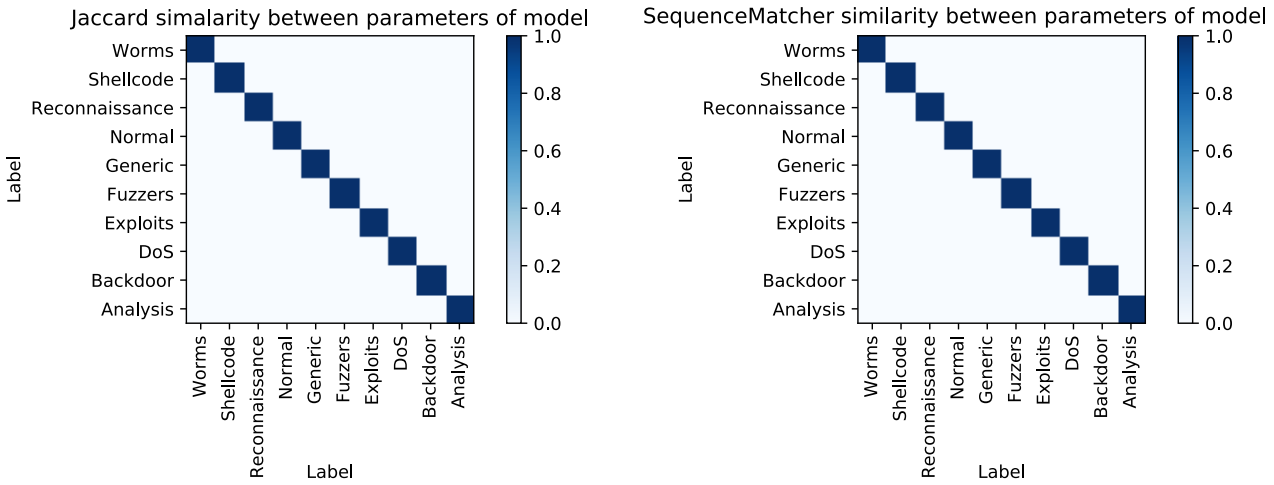
(a) Task $\tau = 5$. (b) Task $\tau = \{2, 3, .., 7\}$

FIGURE 18. Distribution of Ω_k^{μ} of the SI model on the UNSW dataset. Fig. 18a shows the histogram for one task. Fig. 18b shows the histogram evolution through progressive learning. It should be noted that the Ω_k^{μ} are calculated starting from task $\tau = 2$.



(a) EWC (b) FSI

FIGURE 19. Histogram evolution of F_k^{μ} for the EWC model and Φ_k^{μ} for the FSI model on the UNSW dataset.



(a) Jaccard similarity between model weights. (b) SequenceMatcher similarity between model weights.

FIGURE 20. Qualitative similarity between model weights.

3) Fig. 16 shows accuracy evaluations of the Orthogonal Weight Modification (OWM) model using the UNSW and AWID datasets. This figure is also a counterpart of Fig. 2.
4) Fig. 17 shows accuracy evaluations of the Fisher Synaptic Intelligence (FSI) model using the UNSW and AWID datasets. This figure is also a counterpart of Fig. 2. Of note is the similarity between the two figures for the AWID dataset.
5) Fig. 18a shows the distribution of Ω_k^{μ} for task $\tau = 5$ of the UNSW dataset in the SI model. It is another representation of the SI histogram in Fig. 8a.
6) Fig. 18b shows the evolution of the histograms of Ω_k^{μ} as the SI model learns tasks $\tau = \{2, 3, 4, \ldots, 7\}$. Note

the successive broadening of the histogram. This figure complements the information given in Fig. 18a.

7) Fig. 19 shows the evolution of the histograms of F_k^μ for EWC and Φ_k^μ for FSI as they learn tasks $\tau = \{2, 3, 4, \ldots, 7\}$.
8) Fig. 20 shows the qualitative similarity between model weights. The heuristic metrics Jaccard and SequenceMatcher are used to calculate the similarity ratio between adjacent tasks for the UNSW dataset. This figure complements Fig. 13.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their constructive comments on the original version of this manuscript.

REFERENCES

[1] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2935–2947, Dec. 2018.
[2] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowl.-Based Syst.*, vol. 212, Jan. 2021, Art. no. 106622.
[3] I. Guyon, L. Sun-Hosoya, M. Boullé, H. J. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, A. Statnikov, W.-W. Tu, and E. Viegas, "Analysis of the AutoML challenge series 2015–2018," in *Automated Machine Learning: Methods, Systems, Challenges*, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Cham, Switzerland: Springer, 2019, pp. 177–219, doi: 10.1007/978-3-030-05318-5_10.
[4] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," *Proc. Mach. Learn. Res.*, vol. 70, Mar. 2017, pp. 3987–3995. [Online]. Available: http://proceedings.mlr.press/v70/zenke17a.html
[5] K. James, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, and D. Hassabis, "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 13, pp. 3521–3526, Mar. 2017.
[6] G. Zeng, Y. Chen, B. Cui, and S. Yu, "Continual learning of context-dependent processing in neural networks," *Nature Mach. Intell.*, vol. 1, no. 8, pp. 364–372, Aug. 2019.
[7] H. Li, S. Enshaeifar, F. Ganz, and P. Barnaghi, "Continual learning in deep neural network by using a Kalman optimiser," 2019, *arXiv:1905.08119*. [Online]. Available: http://arxiv.org/abs/1905.08119
[8] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6467–6476.
[9] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 139–154.
[10] J. Rajasegaran, M. Hayat, S. Khan, F. S. Khan, L. Shao, and M.-H. Yang, "An adaptive random path selection approach for incremental learning," 2019, *arXiv:1906.01120*. [Online]. Available: http://arxiv.org/abs/1906.01120
[11] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Netw.*, vol. 113, pp. 54–71, May 2019.
[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[13] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.
[14] N. Moustafa and J. Slay, "The significant features of the UNSW-NB15 and the KDD99 data sets for network intrusion detection systems," in *Proc. 4th Int. Workshop Building Anal. Datasets Gathering Exper. Returns Secur. (BADGERS)*, Nov. 2015, pp. 25–31.
[15] R. Kemker, M. McClure, A. Abitino, T. L. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," in *Proc. AAAI*, 2018, pp. 3390–3398.
[16] T. Lesort, H. Caselles-Dupré, M. Garcia-Ortiz, A. Stoian, and D. Filliat, "Generative models from the perspective of continual learning," 2018, *arXiv:1812.09111*. [Online]. Available: http://arxiv.org/abs/1812.09111
[17] J. Xu and Z. Zhu, "Reinforced continual learning," 2018, *arXiv:1805.12369*. [Online]. Available: http://arxiv.org/abs/1805.12369
[18] R. Venkatesan, H. Venkateswara, S. Panchanathan, and B. Li, "A strategy for an uncompromising incremental learner," 2017, *arXiv:1705.00744*. [Online]. Available: http://arxiv.org/abs/1705.00744
[19] R. R. Karn, P. Kudva, and I. M. Elfadel, "Criteria for learning without forgetting in artificial neural networks," in *Proc. IEEE Int. Conf. Cognit. Comput. (ICCC)*, Jul. 2019, pp. 90–97.
[20] *Continual Learning Through Elastic Weight Consolidation Implementation*. Accessed: Aug. 6, 2019. [Online]. Available: https://github.com/yashkant/Elastic-Weight-Consolidation
[21] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," 2012, *arXiv:1211.5063*. [Online]. Available: https://arxiv.org/abs/1211.5063
[22] G. Qian, S. Sural, Y. Gu, and S. Pramanik, "Similarity between Euclidean and cosine angle distance for nearest neighbor queries," in *Proc. ACM Symp. Appl. Comput. (SAC)*, 2004, pp. 1232–1237.
[23] S. Pramanik and K. Mondal, "Cosine similarity measure of rough neutrosophic sets and its application in medical diagnosis," *Global J. Adv. Res.*, vol. 2, no. 8, pp. 212–220, 2015.
[24] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, "Using of Jaccard coefficient for keywords similarity," in *Proc. Int. Multiconf. Eng. Comput. Sci.*, 2013, vol. 1, no. 6, pp. 380–384.
[25] *Python Standard Library—Sequence Matcher*. Accessed: Mar. 6, 2018. [Online]. Available: https://docs.python.org/2/library/difflib.html
[26] G. Golub and C. V. Loan, *Matrix Computations*, 3rd ed. Baltimore, MD, USA: The Johns Hopkins Univ. Press, 1996.
[27] H. R. Loo, S. B. Joseph, and M. N. Marsono, "Online incremental learning for high bandwidth network traffic classification," *Appl. Comput. Intell. Soft Comput.*, vol. 2016, pp. 1–13, Jan. 2016.
[28] *Cosine Wiki*. Accessed: Jun. 30, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Cosine_similarity
[29] J. Plank. *CS494 Lecture Notes—MinHash*. Accessed: Sep. 30, 2021. [Online]. Available: https://web.eecs.utk.edu/~jplank/plank/classes/cs494/494/notes/Min-Hash/index.html
[30] *Sequential Search: Defining and Analyzing the Sequential Search Algorithm*. Accessed: Jun. 30, 2021. [Online]. Available: https://barnasahadotcom.files.wordpress.com/2016/09/similarity.pdf
[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[32] S. Kanai, Y. Fujiwara, and S. Iwamura, "Preventing gradient explosions in gated recurrent units," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 435–444.
[33] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1139–1147.
[34] *UNSW-NB15 Dataset Features and Size Description*. Accessed: Aug. 16, 2017. [Online]. Available: https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-NB15-Datasets/
[35] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. Mil. Commun. Inf. Syst. Conf. (MilCIS)*, Nov. 2015, pp. 1–6.
[36] C. Kolias, G. Kambourakis, A. Stavrou, and S. Gritzalis, "Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 184–208, 1st. Quart., 2016.
[37] *Continual Learning Through Synaptic Intelligence Implementation*. Accessed: Aug. 16, 2018. [Online]. Available: https://github.com/ganguli-lab/pathint
[38] F. Chollet. (2015). *Keras: Deep Learning Library for Theano and Tensorflow*. [Online]. Available: https://keras.io/k
[39] *Continual Learning Through Orthogonal Weights Modification Implementation*. Accessed: Aug. 6, 2020. [Online]. Available: https://github.com/beijixiong3510/OWM
[40] W. McKinney, *Python for Data Analysis: Data Wrangling With Pandas, NumPy, IPython*. Sebastopol, CA, USA: O'Reilly Media, 2012.
[41] *Autokeras*. Accessed: Dec. 16, 2019. [Online]. Available: https://autokeras.com/
[42] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 1946–1956.
[43] *Categorical Encoding Guide to Encoding Categorical Values in Python*. Accessed: Aug. 22, 2017. [Online]. Available: http://pbpython.com/categorical-encoding.html
[44] R. Kozik, M. Choras, and J. Keller, "Balanced efficient lifelong learning (B-ELLA) for cyber attack detection," *J. UCS*, vol. 25, no. 1, pp. 2–15, 2019.
[45] G. Bovenzi, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescape, "A hierarchical hybrid intrusion detection approach in IoT scenarios," in *Proc. GLOBECOM IEEE Global Commun. Conf.*, Dec. 2020, pp. 1–7.

[46] J. Martens, I. Sutskever, and K. Swersky, "Estimating the Hessian by back-propagating curvature," in *Proc. ICML*, 2012, pp. 1783–1790.
[47] C. Bishop, "Exact calculation of the Hessian matrix for the multilayer perceptron," *Neural Comput.*, vol. 4, no. 4, pp. 494–501, Jul. 1992.
[48] F. Dan Foresee and M. T. Hagan, "Gauss-Newton approximation to Bayesian learning," in *Proc. Int. Conf. Neural Netw.*, vol. 3, Jun. 1997, pp. 1930–1935.
[49] S. Ruder, "An overview of multi-task learning in deep neural networks," 2017, *arXiv:1706.05098*. [Online]. Available: http://arxiv.org/abs/1706.05098
[50] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "DISTILLER: Encrypted traffic classification via multimodal multitask deep learning," *J. Netw. Comput. Appl.*, vols. 183–184, Jun. 2021, Art. no. 102985.
[51] J. Bingel and A. Søgaard, "Identifying beneficial task relations for multi-task learning in deep neural networks," in *Proc. 15th Conf. Eur. Chapter Assoc. Comput. Linguistics, Short Papers*, vol. 2. Valencia, Spain: Association Computational Linguistics, Apr. 2017, pp. 164–169. [Online]. Available: http://aclweb.org/anthology/E17-2026
[52] D. Liang and Y. Shu, "Deep automated multi-task learning," in *Proc. 8th Int. Joint Conf. Natural Lang. Process. (Short Papers)*, vol. 2, 2017, pp. 55–60. [Online]. Available: http://aclweb.org/anthology/I17-2010
[53] J. Gideon, S. Khorram, Z. Aldeneh, D. Dimitriadis, and E. M. Provost, "Progressive neural networks for transfer learning in emotion recognition," in *Proc. Interspeech*, Aug. 2017, pp. 1098–1102.
[54] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.
[55] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," 2018, *arXiv:1807.05118*. [Online]. Available: http://arxiv.org/abs/1807.05118
[56] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, M. J. Ben-Tzur, B. Recht, and A. Talwalkar, "A system for massively parallel hyperparameter tuning," in *Proc. Int. Conf. Mach. Learn. Syst.*, Mar. 2020, pp. 1–13.
[57] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.
[58] B. Fleischer, S. Shukla, M. Ziegler, J. Silberman, J. Oh, V. Srinivasan, J. Choi, S. Mueller, A. Agrawal, T. Babinsky, and N. Cao, "A scalable multi TeraOPS deep learning processor core for AI trainina and inference," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 35–36.

RUPESH RAJ KARN received the Ph.D. degree from Khalifa University, Abu Dhabi, United Arab Emirates, under the supervision of Prof. Ibrahim Elfadel. His Ph.D. thesis title was "Machine Learning Methods for the Automated Management of Cloud Computing Workloads." During his Ph.D. degree, he worked as an Intern for three months and a Research Scholar for six months at the IBM T. J. Watson Research Center, NY, USA, under the supervision of Prabhakar Kudva. He is currently a Postdoctoral Fellow in electrical engineering and computer science with Khalifa University. His research interests include the application of AI technologies to cloud management and security. His awards include the Best Paper Award from the UAE Graduate Student Research Conference (twice) and the Best Paper Award from the IEEE Conference on Cognitive Computing, Milan, Italy, in July 2019.

PRABHAKAR KUDVA (Senior Member, IEEE) received the Ph.D. degree in computer science from The University of Utah, in 1995. He was on the Adjunct Faculty of Yale University and Columbia University. He is currently a Research Staff Member with the IBM T. J. Watson Research, Yorktown Heights, NY, USA, where he leads several projects in the areas of enterprise data centers, cloud computing for business intelligence and analytics, PaaS, and CaaS. He has received several IBM awards for High-Value Patents and Outstanding Technical Achievements and the IEEE Region 1 Award for Outstanding Contributions to the Design Automation of Resilient Chips and Systems.

IBRAHIM (ABE) M. ELFADEL (Senior Member, IEEE) received the Ph.D. degree from MIT, in 1993. He is currently a Professor of electrical engineering and computer science at Khalifa University, Abu Dhabi, United Arab Emirates. Prior to his current academic position, he was a Research Staff Member and then a Senior Scientist with the corporate CAD organizations at IBM Research and the IBM Systems and Technology Group, Yorktown Heights, NY, USA, where he was involved in the research, development, and deployment of CAD tools and methodologies for IBM's high-end microprocessors. His current research interests include the IoT platform prototyping, energy-efficient edge and cloud computing, secure IoT communications, embedded digital-signal processing, computer-aided design for VLSI, MEMS, and silicon photonics. He was a recipient of the six Invention Achievement Awards, the one Outstanding Technical Achievement Award, and the one Research Division Award, all from IBM, for his contributions in the area of VLSI CAD. His other awards include D. O. Pederson Best Paper Award from the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, the SRC Board of Directors Special Award for "pioneering semiconductor research in Abu Dhabi," and the Best Paper Award from the IEEE Conference on Cognitive Computing, Milan, Italy, in July 2019. He has served on the technical program committees for several leading conferences, including DAC, ICCAD, ASPDAC, DATE, ISCAS, VLSI-SoC, ICCD, ICECS, and MWSCAS. He was the General Co-Chair of the IFIP/IEEE 25th International Conference on Very Large Scale Integration (VLSI-SoC 2017), Abu Dhabi. He is an Associate Editor of the IEEE Transactions on Very Large Scale Integration (VLSI) Systems.