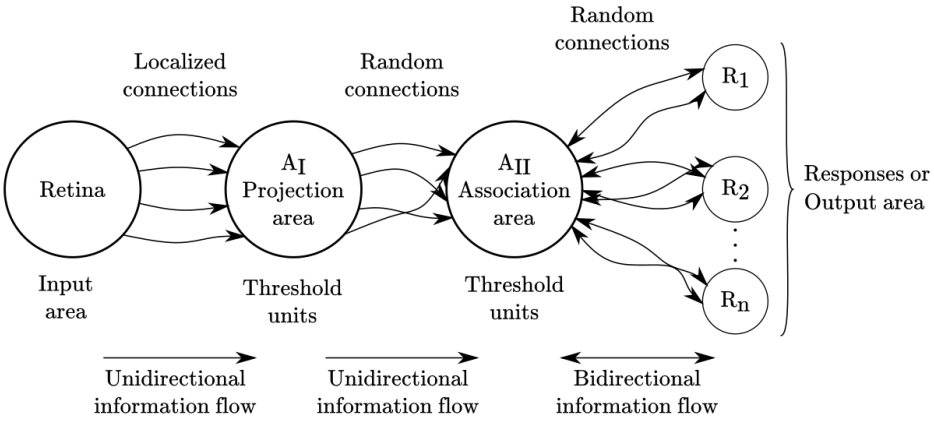
Overcoming Network Saturation in Continual Learning: A Method for Dynamic Parameter Adjustment

MSc. Thesis



Overcoming Network Saturation in Continual Learning: A Method for Dynamic Parameter Adjustment

MSc. Thesis

by

Xusen Qin

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Friday 19 Jan 2024 at 14:00

Student Number:	5594979	
Daily Supervisor:	MSc. Aleksandr Dekhovich	
Project Duration:	Oct 2022 - Jan 2024	
Thesis committee:	Dr. M.H.F. Sluiter,	TU Delft, ME Faculty
	Dr. S. Kumar,	TU Delft, ME Faculty
	Dr. Kevin Rossi,	TU Delft, ME Faculty
	Dr. M.A. Bessa,	Brown University, School of Engineering

TU Delft Delft University of Technology

Keywords: Deep Learning, Continual Learning, Network Saturation, Image recognition, Data-Driven Analysis
Front Cover: Multilayer perceptron, the origin of deep learning. Image Rosenblatt. [1]

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

We can only see a short distance ahead,
but we can see there's plenty to be done.

Alan Turing

Acknowledgements

Academic pursuits are like a journey, where the end of each phase marks the beginning of a new one. Looking back over the years, the beautiful times and scenery are still vivid in my mind. As this chapter comes to an end and a new phase begins, I sincerely thank everyone and everything I have encountered along the way.

I sincerely thank my parents, Mr. Qin Yong and Ms. Zhou Li. It is you who bestowed upon me the gift of life, provided me with unassuming yet warm care, and supported every choice I made at each stage of my life. I am grateful for your companionship through every challenging or joyful moment. Under your watchful care, I have grown from a green youth into a spirited young adult, helping me earnestly search for my direction in the vast ocean of life.

During my master's academic career, I am deeply grateful to Professor Miguel A. Bessa and Professor Marcel H.F. Sluiter. I thank Professor Bessa for his selfless guidance on my project, which helped me make an interdisciplinary transition and opened the door to research in artificial intelligence. I am also grateful to Professor Sluiter for his constructive insights into my research. Additionally, I extend my gratitude to my senior colleague, Aleksandr Dekhovich, for his meticulous assistance with my project. Thank you for your patience and guidance when I was overwhelmed by difficulties.

During my academic journey abroad, I am deeply grateful for the companionship and support of my friends. I thank Wu Jianzhang for our mutual progress in life and studies; I am grateful to Wang Yan for all her help with my graduation process; thank you to Barkav Sudhakar for his assistance in the field of aerospace structures; special thanks to Ge Yipeng, Wen Junhan, and Liu Banxian for their guidance and encouragement in the field of artificial intelligence. I am especially thankful to my beloved Peng Linhan for her devoted companionship. Thank you for being with me through the toughest times. I am incredibly fortunate to have met you in this beautiful country.

Additionally, I am grateful to my four cats, Ms. Flower, Mr. Yello, Ms. Mao, and Mr. Ball, for bringing healing and hope into my life. A special thanks to Ms. Flower for 16 years of companionship. You were there as I grew up, and I was there as you aged. Thank you for being a part of my world.

Looking back on these two years of my academic journey, I am grateful for my own perseverance, for the growth and resilience I have developed in the face of challenges, and for choosing to persist despite numerous temptations to give up. Although there are regrets, I am thankful for my efforts and hard work.

May I return from this journey of a thousand sails, still with the heart of a youth!

Xusen Qin
Delft, Jan 2024

Summary

Neural networks have made significant progress in domains like image recognition and natural language processing. However, they encounter the challenge of catastrophic forgetting in continual learning tasks, where they sequentially learn from distinct datasets. Learning a new task can lead to forgetting important information from previous tasks, resulting in decreased performance on those earlier tasks. This issue is further intensified in dynamic scenarios where the task sequence varies unpredictably. To address this problem, architectural methods have been developed to modify a neural network's structure, creating or adapting subnetworks to retain task-specific knowledge and mitigate catastrophic forgetting. However, these solutions can lead to network saturation, where the accumulation of task-specific adaptations hampers the network's ability to learn new tasks. This research aims to address the problem of network saturation by developing innovative methods that enable neural networks to maintain high performance across both existing and new tasks in continual learning scenarios. Eventually, the new model improved its learning ability on new tasks in the presence of an allowable forgetting, while demonstrating better overall learning ability.

Contents

1

Introduction

In recent years, neural networks have demonstrated exceptional performance in various domains such as image classification, natural language processing, and data regression. However, when deployed in real-life scenarios, deep learning models often encounter the challenge of continual learning. In this context, training data is organized into distinct tasks, with the neural network training for each task separately. For example, this could involve real-time image data streaming from a camera or health monitoring data from a mobile device. In such scenarios, the neural network model is sequentially trained on these task-specific datasets, and it lacks a comprehensive view of all the tasks. Consequently, when learning a new task, the neural network may inadvertently "forget" information from previously learned tasks. This results in the model performing well on the new task but poorly on the older tasks. Furthermore, in dynamic environments, the sequence of tasks may change, further challenging the network's memory capabilities. This decline in neural network performance on previously learned tasks is commonly referred to as "Catastrophic Forgetting" [2, 3].

There are several methods to address the issue of catastrophic forgetting in continual learning scenarios. One of these methods, known as "Architectural methods" [4], dynamically adjusts the neural network's architecture to preserve information from previous tasks within task-specific regions. When training on a new task, the model generates a subnetwork within the architecture or expands the network's capacity to accommodate the requirements of the current task. After completing training on a task, the task-specific parameters for certain tasks are fixed to retain the information from previous tasks. This parameter fixing prevents changes to the information related to old tasks, allowing the neural network to maintain consistent performance on those tasks. Examples of Architectural methods include *SupSup* [5], *SpaceNet* [6], and CP&S [7]. While this method effectively mitigates catastrophic forgetting by preserving parameters, it introduces a challenge: as the task sequence grows, the fixed task-specific regions within the network can gradually occupy the entire network and limit the availability of free parameters for learning

1

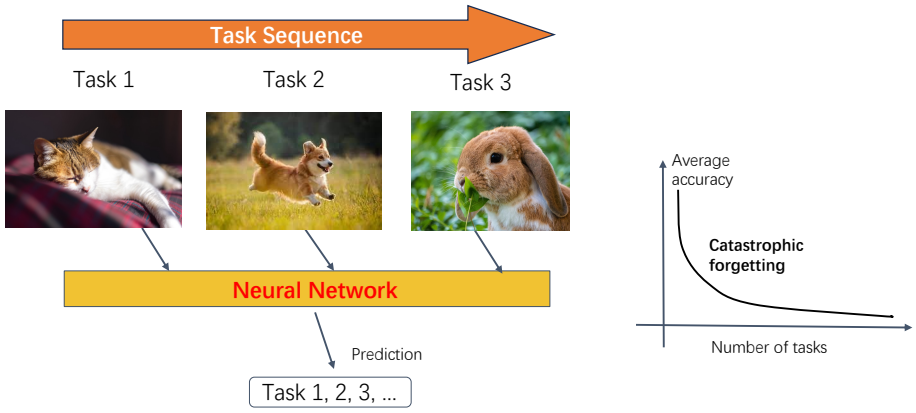


Figure 1.1: Continual learning and catastrophic forgetting in deep learning

new tasks. Consequently, as the task sequence lengthens, the model's capacity to learn new tasks may decrease, leading to a phenomenon known as "Network Saturation" in Continual Learning.

In this study, we introduce the Dynamic Parameters Architectural (DPA) method to address the issue of network saturation in the architectural method of continual learning. DPA dynamically adjusts some of the fixed parameters within the neural network without introducing additional parameters. It ensures that the model can learn new tasks with more parameters by updating certain task-specific parameters, thereby resolving network saturation while maintaining a balance between resistance to forgetting and learning capacity. In this research, we employ Continual Prune-and-Select (CP&S)[7] as the prototypical architectural method to evaluate the efficacy of the DPA. The inclusion of DPA represents an improvement, enhancing the CP&S approach in alleviating network saturation in continual learning. The key innovation lies in DPA's approach to managing parameters within overlapping subnetworks. Unlike CP&S, in which the task-specific parameters are fixed, DPA dynamically updates shared parameters across subnetworks. By freeing shared parameters among subnetworks, DPA is designed to provide a more versatile and adaptable mechanism compared to CP&S. Experimental results are evaluated on both classification and regression datasets, demonstrating that DPA exhibits higher learning efficiency on new tasks. Moreover, in the context of classification problems, DPA effectively mitigates forgetting while improving learning performance, resulting in superior overall learning performance.

In this thesis, Chapter 2 presents a comprehensive review of the state-of-the-art in continual learning, highlighting the existing knowledge gaps that impede solving the problem of network saturation. Chapter 3 evaluates the effectiveness of the DPA methods within the context of classification tasks in continual learning scenarios. Chapter 4 examines the DPA methods' performance in the continual learning framework for data-driven analysis of composite hyperelastic properties. Finally, Chapter 5 summarizes the findings and discusses the advantages of DPA

methods over the conventional CP&S approach.

2

Literature review

As deep learning gains prominence within the realm of artificial intelligence, the issue of *catastrophic forgetting* [8, 3] has emerged as a pressing challenge that demands attention from researchers. This literature review aims to provide an in-depth exploration of the foundational concepts and context surrounding continual learning. Additionally, it delves into the current landscape of state-of-the-art algorithms within the field, typically classified into three main categories: regularization-based, rehearsal-based, and architectural methods. Within our discussion of these algorithms, We intend to highlight the advantages and disadvantages of these methods. We aim to identify important gaps in knowledge within the current research literature through this assessment. These identified gaps will help guide our research objectives and contribute to the ongoing pursuit of solutions to the problem of catastrophic forgetting.

2.1. Catastrophic forgetting

Learning can be perceived as the modification of existing connections and the formation of new ones between neurons [9]. Through parameter adjustments, artificial neural networks can emulate biological neural networks, thus acquiring capabilities to learn, remember, and predict [10]. However, unlike the human brain which can continually adapt to new information, artificial neural networks face a challenge. When exposed to new data, they often lose previously learned knowledge. This phenomenon, known as *catastrophic forgetting*[3, 8], stands as a notable constraint in certain neural network applications.

Catastrophic forgetting occurs when a neural network significantly loses retained knowledge from prior training as it learns from new data streams. This challenge is a fundamental issue in deep learning, constraining the efficacy of neural networks in sequential or incremental learning contexts [3]. For instance, when a deep learning model processes multiple data batches (referred to as 'tasks') in parallel, it maintains consistent learning and predictive accuracy across all tasks. In contrast, when these

batches are introduced sequentially, the model tends to prioritize the most recent task, diminishing its recollection of earlier ones. Consequently, as tasks accumulate, the model's overall performance declines due to its tendency to prioritize the most recent information.

2

One of the main reasons for catastrophic forgetting is the disparate parameter distributions within the neural network across various tasks [2]. In multi-task learning, the dataset for tasks is presented to the model concurrently, leading to a convergence in the neural network's weight distribution once the training loss function attains its minimum [3]. Contrarily, in a continual learning setting, tasks are not introduced to the model simultaneously. Typically, neural networks employ Gradient Descent (GD) to refine network parameters. However, GD primarily emphasizes the gradient of the present data, neglecting insights from prior data [3, 2]. When the model undergoes training for a new task, the loss function's update trajectory for the fresh task aligns with its global optimal direction. Yet, this direction might diverge from the ideal update path for loss functions associated with previous tasks. This divergence leads to parameter shifts in older tasks to minimize the fresh task's loss function, diminishing performance in earlier tasks, and inducing catastrophic forgetting. To counteract this and bolster deep learning's efficacy in sequential task learning, approaches like continual or lifelong learning have been advocated [8, 11]. Figure 2.1 offers a schematic representation of catastrophic forgetting.

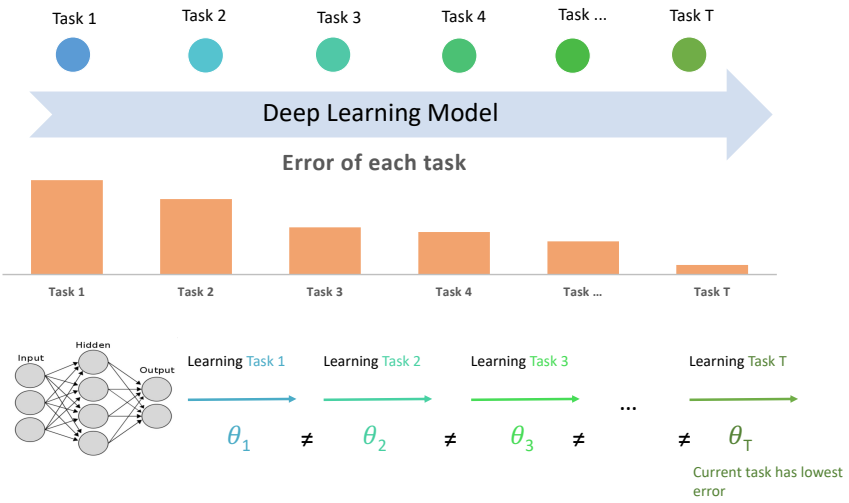


Figure 2.1: The schematic diagram of catastrophic forgetting in deep learning

2.2. Continual learning approaches

Continual learning, often referred to as lifelong learning, pertains to a model's ability to adapt to a continuous sequence of diverse tasks [12]. Usually, these new tasks exhibit minimal or no correlation with preceding ones. The main goal of continual learning techniques is to mitigate catastrophic forgetting, ensuring that machine learning models not only excel in new tasks but also maintain proficiency in earlier ones. Continual learning can be broadly classified into two distinct types [4]:

- **Continual Learning with Task ID**: In this scenario, the identity of the task (task ID) is provided when data is fed into the model during both the training and inference stages. Although it represents a more straightforward continual learning challenge (given the explicit knowledge of the task for test data points), many research works consider this scenario for real-world applications [13, 14]. It is also commonly referred to as task incremental learning (task-IL) [15].
- **Continual Learning without Task ID**: Here, the task ID is supplied only during the training phase and is absent during inference. In practice, this constraint often renders this scenario more complex compared to its counterpart, due to the reduced contextual information about test data [16, 17, 18]. Some methodologies attempt to predict the task ID for test data, though this can introduce new challenges and limitations [19, 20]. When models in this setting are geared towards solving classification problems, it's termed class-incremental learning (class-IL) [15].

In the early stages of continual learning research, temporary buffers were employed to retain data from preceding tasks, allowing models to replay this buffered data and thereby reinforce their acquired knowledge, offsetting catastrophic forgetting to some extent [21]. Modern approaches to continual learning are primarily categorized into three methods: **architectural**, **regularization**, and **rehearsal** [4]. **Architectural methods** dynamically refine the neural network by either appending additional parameters [22] or by formulating task-specific parameters to encapsulate information from earlier tasks [23, 20]. **Regularization methods** leverage various strategies, such as harnessing Bayesian properties inherent to incremental learning [24, 25, 12], restricting alterations in network parameters [26, 27], or employing knowledge distillation when there's a significant parameter shift, ensuring the conservation and transfer of insights from prior tasks [16]. Meanwhile, **rehearsal methods** emphasize retaining and replaying past task data during the training of subsequent tasks [17, 28, 29]. A visual representation of the latest advancements in continual learning can be found in Figure 2.2.

2.3. Rehearsal methods

Humans frequently reinforce their memories through revisitation. This process is similar to students preparing for exams or athletes refining specific moves. Neural networks can emulate this human-like review approach through what is termed

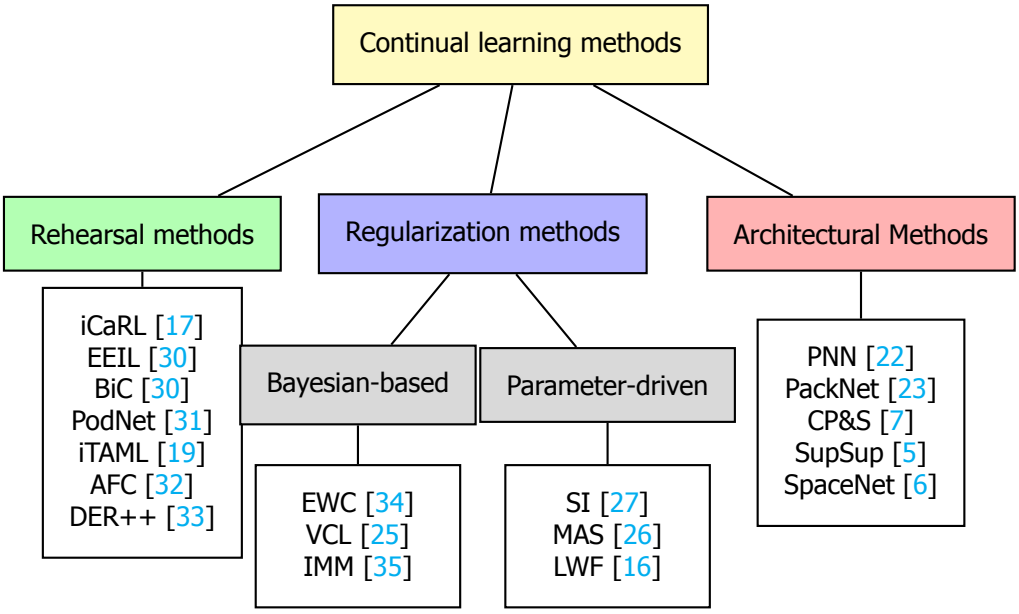


Figure 2.2: Different researches on continual learning.

"rehearsal methods" [15]. A significant portion of contemporary continual learning research employs these methods, with notable examples including iCaRL [17], EEIL [30], BiC [30], PodNet [31], iTAML [19], AFC [32] and DER++ [33]. However, these techniques often require dedicated storage, leading to increased memory usage by algorithms. Moreover, due to privacy concerns, previous task data may become inaccessible when a new task is introduced [36]. Therefore, this research will not explore rehearsal methods in greater detail.

2.4. Architectural methods and network pruning

Human memory is a multifaceted mechanism. The remarkable capacity of organisms to retain information is attributed to the brain's versatility in storing diverse data across its regions [37]. To emulate the nuanced memory systems found in human brains, architectural methods in machine learning have been developed. These methods counteract catastrophic forgetting by incorporating strategies such as: introducing new branches for incoming tasks to prevent parameter overwriting [22], segregating information across distinct model replicas [38], or allocating information to different segments of a neural network [39, 23, 20].

Progressive Neural Networks (PNN or progressive network) [22] represent a notable approach grounded in the dynamic architecture method customized for continual learning scenarios. In PNNs, older parameters undergo refinement during training, ensuring retention of insights from preceding tasks (refer to appendix A.1 for details). Nonetheless, the implementation of PNNs necessitates additional

storage to preserve the finalized model for each distinct task. Consequently, with an increasing number of tasks, the parameter count also grows, placing additional demand on the device's memory capacity.

Maintaining memory by storing an entire neural network model proves to be both inefficient and burdensome in terms of memory consumption. Therefore, a viable solution to this challenge is to allocate distinct subnetworks within the main network to handle different tasks. Once training on a particular task concludes, the parameters of its corresponding subnetwork are solidified, ensuring no future modifications. During the evaluation, the appropriate subnetwork is engaged to process the test data. Techniques like iterative pruning can be employed to establish these subnetworks [40].

Network pruning is a widely-used technique for compressing neural networks, aiming to create a more compact yet efficient model by eliminating less significant parameters related to a specific task [41]. Researchers emphasize the presence of numerous redundant neurons and weights within the neural network structure, providing the theoretical basis for its compression [42]. In the context of continual learning, several network pruning methodologies have emerged in recent times [23, 20, 39].

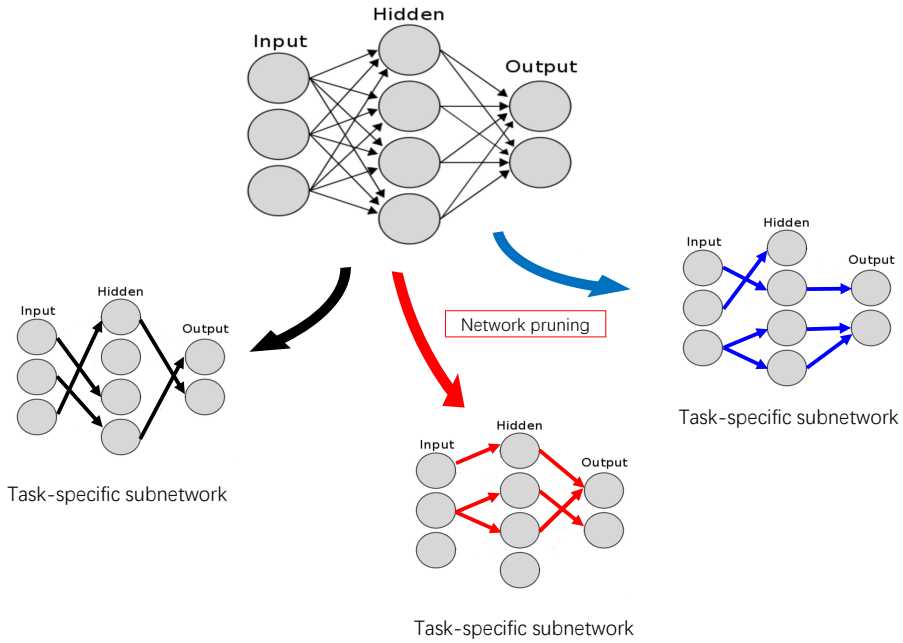


Figure 2.3: The schematic diagram of creating a task-specific network by network pruning

PackNet leverages network pruning techniques to free parameters, enabling the assimilation of new tasks without expanding the network's capacity [23]. The foundational concept behind PackNet is to initiate a large-scale neural network. As

a task is introduced, the model dedicates a subset of its parameters to this input. Once training concludes for the task, these allocated parameters become immutable (as detailed in Appendix A.2). Nonetheless, there is an inherent challenge: as tasks consume the available parameters, the network eventually saturates. This saturation implies that without expanding its parameter storage, the model can't accommodate additional tasks. Furthermore, PackNet's reliance on a rudimentary magnitude-based pruning method [40] does not yield a sufficiently sparse network representation. It's also worth noting that PackNet operates optimally in continual learning scenarios where the task ID is provided during inference. These limitations constrict the broader adaptability of PackNet.

To address challenges in precise model pruning, task-specific parameter compression, and the enhancement of neural network performance in continual learning scenarios without explicit task IDs, the **Continual-Prune-and-Select (CP&S)** method is introduced [20]. In contrast to PackNet, which identifies subnetworks, CP&S employs iterative pruning to discover task-specific subnetworks with strong learning capabilities. Once optimized for a task, parameters within these subnetworks are solidified. Interestingly, CP&S allows for subnetwork overlap, enabling them to share parameters, regardless of their mutability status. This overlap represents the merging of newly formed subnetworks with previous ones. When new task data is introduced, both mutable and immutable parameters within the subnetwork are utilized for forward propagation, but only mutable parameters receive gradient updates. This mechanism ensures the new subnetwork imbibes insights from previous tasks without overwriting their established parameters. For inference, when provided a task ID, CP&S activates the corresponding subnetwork. In scenarios lacking a task ID, CP&S predicts it, initiating the relevant task-specific subnetwork. Due to the unchangeability of task-specific parameters, each subnetwork consistently retains its post-training performance..

The CP&S training includes both pre-training and re-training phases. During pre-training, CP&S identifies a subnetwork within the model by evaluating the *importance score* (IS) [7] for each connection, enabling the creation of task-specific masks and parameter assignments for tasks. For layer l, the input is represented by the pruning set $X^{(l-1)} = \{x_n^{(l-1)}\}_{n=1}^{N}$, where each element $x_n^{(l-1)} = \{x_{nk}^{(l-1)}\}_{k=1}^{m_{l-1}}$ has a dimension of m_{l-1}. The contribution from a connection between neuron i in layer $l-1$ and neuron j in layer l is given as the average output $\frac{1}{N}\Sigma_{n=1}^{N}\left|w_{ij}^{(l)} x_{ni}^{(l-1)}\right|$. The importance score of the bias for neuron j in layer l is taken as the bias value $\left|b_j^{(l)}\right|$ relative to the total output of layer l. This is illustrated in Fig.2.4 and formally expressed in Eq.2.1.

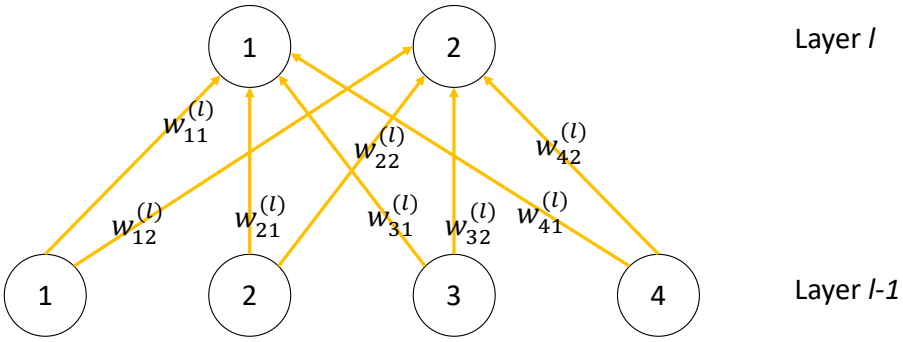


Figure 2.4: Definition of importance score for each connection and bias is determined by the proportion of the neuron's output as well as values of bias to the total output of layer l [7].

$$\text{For weights: } s_{ij}^{(l)} = \frac{\frac{1}{N}\Sigma_{n=1}^{N}\left|w_{ij}^{(l)}x_{ni}^{(l-1)}\right|}{\Sigma_{k=1}^{m_{l-1}}\left(\frac{1}{N}\Sigma_{n=1}^{N}\left|w_{kj}^{(l)}x_{nk}^{(l-1)}\right|\right)+\left|b_j^{(l)}\right|}$$
$$\text{For bias: } s_{bias,j}^{(l)} = \frac{\left|b_j^{(l)}\right|}{\Sigma_{k=1}^{m_{l-1}}\left(\frac{1}{N}\Sigma_{n=1}^{N}\left|w_{kj}^{(l)}x_{nk}^{(l-1)}\right|\right)+\left|b_j^{(l)}\right|} \tag{2.1}$$

The importance score for each connection serves as a benchmark for the pruning function, determining which connections should be pruned prior to processing through layer l. A hyperparameter, α, is set such that if the cumulative importance score falls below $(1-\alpha)\sum_{i=1}^{m} s_{ij}$, then the connection is eliminated from the network during the learning of a specific task (refer to the pseudo-code in Algorithm.1). Given a sequential dataset $D_{1:T} = \{D_i\}_{i=1}^{T}$ in a continual learning context, the pruning function generates a series of masks $M_{1:T} = \{M_t\}_{t=1}^{T}$, each corresponding to an individual task. Every mask, denoted as $M_t = \left\{m_{ij}^t\right\}_{i,j}$, establishes a subnetwork tailored to a particular task.

2

Algorithm 1 Pseudo-code for pruning function in CP&S

Require: network $\mathcal{N}$, training dataset X, pruning hyperparameter α.

$X^{(0)} \leftarrow X$
for every layer $l = 1, \ldots, L$ **do**
  $X^{(l)} \leftarrow \text{layer}(X^{(l-1)})$
  for every neuron j in layer l **do**
    Compute importance scores $s_{ij}^{(l)}$ for every connection $w_{ij}^{(l)}$ and bias $b_j^{(l)}$ using Eq. 2.1.
    $\hat{s}_{ij}^{(l)} \leftarrow \text{Sort}(s_{ij}^{(l)}, order = descending)$.
    Find $p_0 = \min\{p : \sum_{i=1}^{p} \hat{s}_{ij}^{(l)} \geq \alpha\}$.
    Prune connections with importance score $s_{ij}^{(l)} < \hat{s}_{p_0 j}^{(l)}$.
  end for
end for

Fig. 2.5 presents the performance of CP&S on the ImageNet-100 dataset [43]. When compared to other state-of-the-art continual learning methods, CP&S demonstrates superior performance when the ImageNet-100 dataset is divided into 10 tasks. CP&S maintains a consistently high Top-5 classification accuracy when optimized using Adam with a batch size (bs) of 20, showing negligible decrement. This can be attributed to the stabilization of the shared segment among subnetworks, ensuring the preservation of parameter distributions for specific tasks. On the CIFAR-100 dataset [44], CP&S consistently exhibits commendable accuracy rates along with minimized forgetting rates. Remarkably, when the dataset is divided into 5 or 10 tasks, CP&S's performance remains consistent, regardless of the availability of the task ID, highlighting the model's ability to accurately determine task affiliations (as depicted in Fig. 2.6).

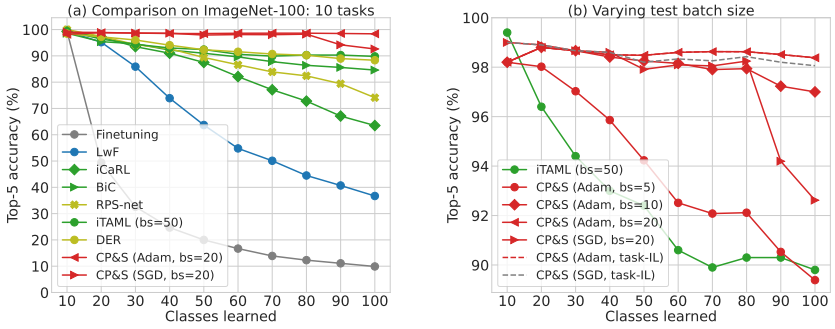


Figure 2.5: The performance comparison of CP&S in ImageNet-100 [43] split in 10 tasks [20]. The bs refers to batch size.

However, when the task number increases to 20, the accuracy drops significantly. This happens because as soon as a parameter is assigned to a task, it can never be updated with gradient descent. Therefore, there are fewer available train-

ing parameters after every newly learned task. This phenomenon is called *network saturation* and is common to all architectural methods based on subnetworks within a fixed-size network.

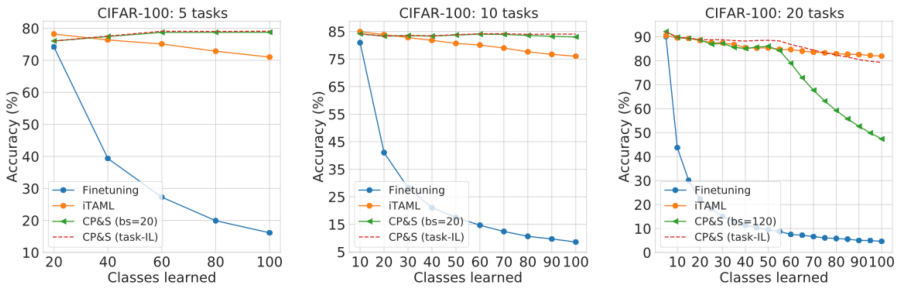


Figure 2.6: The performance comparison of CP&S in CIFAR-100 [44] split in 5, 10 and 20 tasks [20].

In architectural methods, the challenge of catastrophic forgetting has primarily been addressed by fixing parameters in the overlapping regions between the current and preceding subnetworks. However, this strategy of freezing parameters can hinder the model's adaptability when exposed to new tasks. The integration of knowledge from prior tasks can enhance learning efficiency for newer tasks by leveraging previously utilized task-specific parameters [45]. When parameters remain static, training for a new task becomes isolated from historical tasks. Thus, the central aim of this research is to refine the learning paradigm of architectural methods for new tasks. This strives to achieve minimal forgetting without an increase in the parameter count for previous tasks. One potential solution to address parameter saturation is to moderate the constraints on the subnetwork overlap and update it using insights from the new task. Finding a balance between retaining old data and assimilating new data becomes crucial. Additionally, harnessing the reusability of parameters can enhance the model's learning capability for fresh tasks. Given the preference against direct retention of historical data, the deployment of regularization methods becomes essential to confine updates in the shared parameters of the architectural method.

2.4.1. Dynamic parameters architectural methods

Since the task-specific parameters in the architectural methods are fixed to prevent the change in training the new task, these parameters cannot assist subsequent learning. To improve the reusability of parameters, the dynamic parameters architectural methods are proposed. During the training process, the dynamic parameters architectural method first adjusts the neural network architecture in the continual learning scenario and creates task-specific parameters for each task. Different from the architectural method that freezes the specific parameters for the previous tasks, the main idea for this implementation is to update part of unimportant previous parameters in training on the new task [46, 6]. By reusing the previous parameters, training on new tasks will be less dependent on the free weights in the neural network, and the requirement of storing extra parameters also reduces.

Meanwhile, old information can be transferred to the new task learning through the sharing parameters [45]. By reusing the parameters, the old parameters can be used to improve the learning performance of the model on the new parameters. The recent research on dynamic parameters architectural methods are as follows.

Dynamically Expandable Network (DEN) [46] augments the number of trainable parameters when introducing a new task, thereby expanding the neural network's information capacity. DEN operates in an online fashion, employing selective retraining to enhance network capacity. Contrasting with PackNet [23], neuron count determination is driven by group sparse regularization instead of utilizing the entirety of neural network parameters. Concurrently, a subset of parameters corresponding to prior tasks is chosen for updating during new task training. This is achieved by constructing sparse connections within the model, with parameters outside these connections remaining static. A schematic representation of DEN can be viewed in Fig.2.7. In a subsequent development, **Bayesian compression for dynamically expandable networks (BCDEN)** [47] employs Bayesian techniques to streamline the model's architecture while preserving its accuracy. This results in fewer neurons being updated across each hidden layer when new tasks are introduced, leading to the solidification of more parameters during updates.

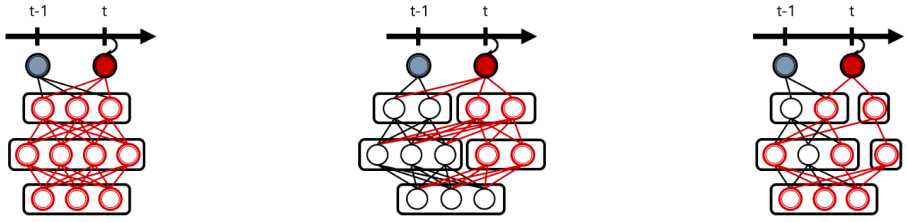


Figure 2.7: An overview of Dynamically Expandable Network (DEN) for learning in the continual learning scenario [46]

Feature Boosting and Compression (FOSTER) [48] enhances dynamic architectures by mapping expanding feature vectors to trainable linear layers of fixed dimensions. This method enlarges the new model to accommodate the residuals of the old model and employs knowledge distillation to update the parameters that are shared between the new model and previous iterations.

Dynamically adjusting task-specific parameters during training on new tasks can mitigate the capacity constraints of the neural network. However, a common limitation in contemporary research on dynamic parameter architectural methods is the frequent expansion of the model size. This continual growth of parameters can place significant strain on device memory. Consequently, devising strategies to update parameters without further inflating the model's size emerges as a paramount challenge warranting further investigation in this field.

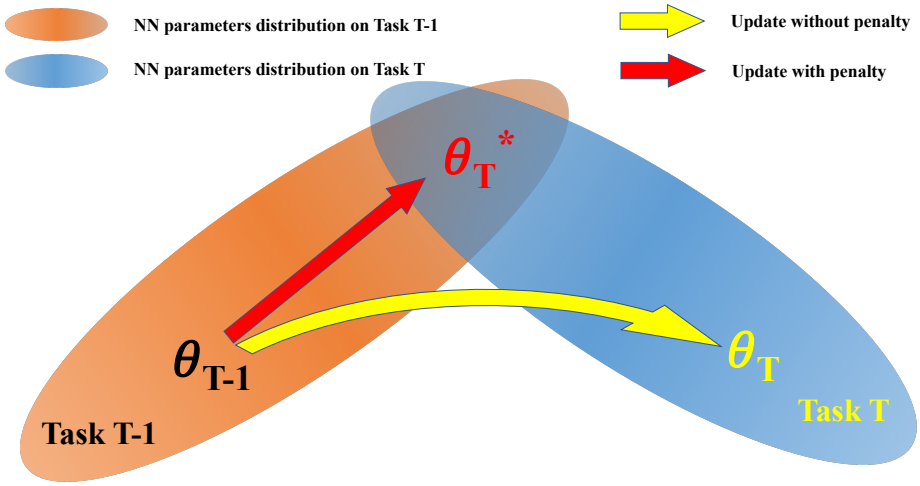


Figure 2.8: The neural network updated on the new task will be based on the optimal parameter space of previous tasks. If the neural network is updated without penalty (yellow arrow), the parameters will leave the optimal space for old tasks (orange region) and enter the optimal space for the new task (blue region). When the parameters are updated with the penalty (red arrow), the new parameters will converge to the optimal parameters space on both the old tasks and the new task (overlap between orange and blue regions). [24]

2.5. Regularization methods

In continual learning, tasks arrive at the neural network in batches. Let $D_T = \{X_n, Y_n\}_{n=1}^{N}$ be the data collection of task T. The performance of neural networks on different tasks can be considered as the performance of the model in the joint dataset $D_{1:T} = \{D_t\}_{t=1}^{T}$. The neural network should not only learn the data D_T of task T but also have a good performance on the previous tasks from D_1 to D_{T-1} to prevent catastrophic forgetting.

One approach to counteract catastrophic forgetting is to constrain the updates to parameters initiated by the introduction of a new task, thereby preventing significant alterations [49]. Parameter updates are fundamentally tied to the minimization of the current task's loss function. By incorporating a penalty related to prior tasks into the loss function for regularization purposes, the network's updates are connected to historical tasks. Consequently, substantial deviations in parameters are deterred by this regularization term. As illustrated in Fig. 2.8, the basic tenet of regularization methods is depicted: the orange region demarcates the parameter space associated with previous tasks (representing the low error zone for tasks $1 : T-1$), while the blue region delineates the parameter space pertinent to new tasks (signifying the low error zone for task T). If the neural network operates using parameters from historical tasks and simultaneously trains on a new task, the trajectory of parameter optimization is represented by the yellow arrow. The parameters gravitate towards the low error region of the new task, subsequently deviating from the optimal parameter realm of bygone tasks. However, if regu-

larization methods are employed (as indicated by the red arrow), the parameters are steered towards the intersection of the parameter spaces of both the old and current tasks. After the completion of training for task T, the model retains traces of earlier tasks, indicating that the neural network demonstrates good performance across both historical and current tasks.

In many cases, data from the previous tasks is not available when a new task arrives. Instead of keeping the data in external storage, regularization methods introduce a regularization term in the loss function during the training process to solidify the parameters updated with previous tasks [4]. The regularization-based loss functions are usually expressed as follows:

$$\mathcal{L}_{total} = \mathcal{L}(\hat{Y}_T, Y_T) + \lambda \cdot \Omega \cdot ||\theta - \theta_{1:T-1}|| \tag{2.2}$$

The total loss function, denoted as $\mathcal{L}_{total}$ and regulated by Eq.2.2, is employed to direct the updates in the neural network. In this equation, Y_T represents the ground truth label of Task T, while $\hat{Y}_T$ signifies the label output by the network during the training process. The weight matrices of the model for task T and for prior tasks (from task 1 to task $T-1$) are given by θ and $\theta_{1:T-1}$, respectively. The term λ acts as a hyper-parameter, determining the impact of the regularization term. The matrix Ω, which captures the importance of each parameter's change, is crucial in constraining the parameter distribution. The computation of Ω can be categorized into Bayesian-based methods and Parameter-driven methods.

2.5.1. Bayesian based methods

The output of a neural network model can be considered as a probability distribution function (pdf) $P(D|\theta)$ [50]. D is the data that is trained and θ is the parameters of the neural network. The parameters can be inferred by Maximum Likelihood Estimation (MLE):

$$\theta^{MLE} = \arg\max_{\theta} \log P(D|\theta) \tag{2.3}$$

In MLE, the prior distribution of the $P(\theta)$ is not considered, which means the distribution $P(\theta)$ has no relevancy to the model parameters structure. However, in the continual learning scenario, the update of the neural network on the new tasks is based on the optimal parameters of previous tasks. So, the previous parameters can be considered as the prior of the new parameters. In the Bayesian perspective, the Maximum Posteriori (MAP) of the parameters can be calculated by the likelihood $P(D|\theta)$ and the prior of the parameters $P(\theta)$. And by taking the logarithm of the likelihood, we can get the loss function $\mathcal{L}(D)$ [51]. The MAP of model parameters θ is described as follows:

$$\begin{aligned}\theta^{MAP} &= \arg\max_{\theta} \log P(\theta|D) \\ &= \arg\max_{\theta} \log P(\theta|D) + \log P(\theta) - \log P(D) \\ &= \arg\max_{\theta} \log P(D|\theta) + \log P(\theta) \\ &= \arg\min_{\theta} \mathcal{L}(D) - \log P(\theta)\end{aligned} \tag{2.4}$$

In the case of continual learning, the model receives the data in batches $D_1, D_2, \ldots, D_T$. We define the overall dataset as $D_{1:T} = \{D_i\}_{i=1}^{T}$. If we consider the learning process from task $T-1$ to task T, we can write the overall dataset as a set from already observed data $D_{1:T-1}$ and the new data D_T:

$$D_{1:T} = \{D_{1:T-1}, D_T\} \tag{2.5}$$

The probability distribution function (pdf) in continual learning of neural network parameters is $P(\theta|D_{1:T})$. Owing that the θ is inferred based on the posterior distribution of the previous parameters, $P(\theta|D_{1:T-1})$ is the proiri of the $P(\theta|D_{1:T})$. As the MAP of parameters from Eq.2.4 and Eq.2.5 can be changed into the following form:

$$\begin{aligned}\theta^{MAP} &= \arg\max_{\theta} \log P(\theta|D_{1:T}) \\ &= \arg\max_{\theta} \log P(D_T|\theta) + \log P(\theta|D_{1:T-1}) \\ &= \arg\min_{\theta} \mathcal{L}(D_T) - \log P(\theta|D_{1:T-1})\end{aligned} \tag{2.6}$$

Eq.2.6 represents an adaptation of Bayes' rule tailored for a continual learning context. This foundational equation illuminates the essence of Bayesian inference within neural networks throughout continual learning. It underscores the idea that when the model is introduced to new tasks, its parameters' updates should be anchored and influenced by previously learned parameters. As a result, the past knowledge embedded within the previous distributions is retained during subsequent training phases. The challenge, however, lies in accurately capturing the distribution $P(\theta|D_{1:T-1})$. While it's the linchpin to inferring the neural network's optimal parameters, in many scenarios, this posterior distribution eludes a straightforward closed-form solution [52]. A pivotal concern in Bayesian continual learning then becomes the expedient and precise computation of this posterior distribution. A potential solution emerges from the use of exponential family distributions. Specifically, one could employ $q(\theta|\phi_T)$, which provides a closed-form solution characterized by parameters ϕ_T. The intent here is to utilize this distribution as an approximation for $P(\theta|D_{1:T-1})$ [53]. Consequently, the overarching objective crystallizes: it's about discerning that specific set of parameters, ϕ_T, which offers a robust approximation to our desired posterior distribution.

$$q(\theta|\phi_T) \approx P(\theta|D_{1:T-1}) \tag{2.7}$$

The Laplace approximation is a simple and widely used method for estimating probability distributions approximation [54]. The basic idea of the Laplace approximation is to use a Gaussian distribution with parameters $\phi = (\mu, \sigma)$ to approximate a complex distribution. If we compute the second order Taylor expansion of $\log P(\theta|D_{1:T-1})$ at $\theta = \theta_{1:T-1}$ then we get the expression of $P(\theta|D_{1:T-1})$ with the expectation is the model parameters that converge on the previous tasks and the Hessian matrix of the old parameters pdf, and the variation is the reciprocal of secondary derivative (Hessian matrix $\mathcal{H}$) of the distribution of the parameters, which is shown in Eq.2.9. Appendix.A.3 illustrates the detail of Laplace approximation.

$$\log P(\theta|D_{1:T-1}) = \log P(\theta_{1:T-1}|D_{1:T-1}) + \frac{1}{2} \cdot \underbrace{\frac{\partial^2 \log P(\theta|D_{1:T-1})}{\partial \theta^2}\Big|_{\theta_{1:T-1}}}_{Hessian} (\theta - \theta_{1:T-1})^2 \tag{2.8}$$

$$P(\theta|D_{1:T-1}) \sim \mathcal{N}\left(\theta_{1:T-1}, \left(-\frac{\partial^2 \log P(\theta|D_{1:T-1})}{\partial \theta^2}\Big|_{\theta_{1:T-1}}\right)^{-1}\right) \tag{2.9}$$

However, computing the second-order derivatives of a neural network with respect to its parameters can be challenging. In a recent development, the **Elastic Weight Consolidation (EWC)** method was introduced, offering a solution to this problem by substituting the Hessian matrix with the Fisher information matrix [34]. It's worth noting that the Hessian matrix, $\mathcal{H}_{\log P(\theta|D_{1:T-1})}$, satisfies the Jacobian of the second derivative of the log-likelihood. By taking the expectation over the Hessian matrix, it becomes apparent that the expectation of $\mathcal{H}$ equates to the diagonal of the Fisher information matrix $\mathcal{F}$ [55]. With the replacement of the Hessian matrix in Eq.2.9, the probability density function (pdf) of $P(\theta|D_{1:T-1})$ can be represented as:

$$P(\theta|D_{1:T-1}) \sim \mathcal{N}\left(\theta_{1:T-1}, \mathcal{F}^{-1}\right) \tag{2.10}$$

Fisher information matrix is defined as the expectation of the outer product of the log first derivative [55]:

$$\mathcal{F} = \mathbb{E}\Big[\nabla \log P(\theta|D_{1:T-1}) \cdot \nabla \log P(\theta|D_{1:T-1})\Big] \tag{2.11}$$

The MAP process of θ in EWC is finally given:

$$\begin{aligned} \theta^{MAP} &= \arg\max_{\theta} \log P(\theta|D_{1:T}) \\ &= \arg\max_{\theta} \log P(D_T|\theta) + \log P(\theta|D_{1:T-1}) \\ &= \arg\min_{\theta} \mathcal{L}(D_T) + \frac{\lambda}{2} \cdot \mathcal{F} \cdot (\theta - \theta_{1:T-1})^2 \end{aligned} \tag{2.12}$$

The total loss function $\mathcal{L}_{EWC}$ in EWC can be written as follows [24]:

$$\underbrace{\mathcal{L}_{EWC}(D_T)}_{\text{Total loss}} = \underbrace{\mathcal{L}(D_T)}_{\text{loss on Task T}} - \underbrace{\frac{\lambda}{2} \cdot \mathcal{F} \cdot (\theta - \theta_{1:T-1})^2}_{\text{regularization term}} \tag{2.13}$$

The Eq.2.13 is the final expression of the loss function when updating the neural network on the Task T; $\mathcal{L}(D_T)$ is the loss function on task T, which is the cross-entropy (classification problem) or MSE (regression problem). Hyperparameter λ that controls the proportion of the regularization term to the overall update process, $\mathcal{F}$ is the new diagonal of the Fisher information matrix and $\theta_{1:T-1}$ is the old parameters that had been trained on previous T-1 tasks. Considering that the implementation of EWC is very effective, it can be used to update the shared parameters between old models and the new model in architectural methods.

Variational continual learning (VCL) [56] and **Incremental Moment Matching (IMM)** [35] are two recent researches solve the approximation of $P(\theta|D_{1:T})$ by other Bayesian methods. Variational continual learning (VCL) applies variational inference [25] in the continual learning scenario: by artificially introducing an exponential family distribution $q(\theta|\phi) \sim \mathcal{N}(\phi)$ with the parameter ϕ to fit the unsolvable distribution $P(\theta|D)$ [56]. While Incremental Moment Matching (IMM) solves the problem of distribution comparison by using moment matching [57] in continual learning. Details are shown in Appendix.A.4 and Appendix.A.5.

2.5.2. Parameter-driven methods

Importance estimation. Not all connections and parameters are necessary for the neural network [42]. If one parameter is significant for the task, the magnitude of updates for it should be significantly limited. How to estimate the moving scale of each parameter without using extra memory is determined by the importance term Ω of each parameter. One paper proposes **Synaptic Intelligence (SI)** [27] to alleviate the catastrophic forgetting issue in neural networks by judging the importance of parameters through the trajectories integral to the training process. The synapse here refers to the "strength" of the interconnection between neurons of the neural network. To some extent, analyzing the importance of the is equivalent to the "perception" of the "synaptic" to its own contribution.

The parameters update during the training in each epoch can be expressed as the multiply of minor migration of parameters $\delta(t)$ and the gradient $g = \frac{\partial \mathcal{L}}{\partial \theta}$ in terms of the change of loss function ($\mathcal{L}$) in dataset D:

$$\mathcal{L}(D, \theta(t) + \delta(t)) - \mathcal{L}(D, \theta(t)) \approx \sum_{i,j} g_{ij}(t)\delta_{ij}(t) \tag{2.14}$$

The minor migration of parameters $\delta_{ij}(t)$ can be considered as the derivative of parameters (θ_{ij}) with respect to t, which is $\delta_{ij}(t) = \frac{\partial \theta_{ij}}{\partial t}$. Since $g_{ij}(t)$ and $\delta_{ij}(t)$ are both the derivative of parameters:

$$\int_{t_{T-1}}^{t_T} g(\theta(t)) \cdot \delta(t) d\theta = \sum_{i,j} \int_{t_{T-1}}^{t_T} g_{ij}(t)) \delta_{ij}(t) dt = -\sum_{i,j} \omega_{i,j}^T \tag{2.15}$$

The path integral $\omega_{i,j}^T$ can illustrate the contribution of each parameter to the total loss in task T, which can be simply approximated as the product with the gradient $g_{ij}(t)$ with the parameters update $\delta_{ij}(t)$. In the training process, the updated scale of the parameter θ_{ij} is related to its path integral $\omega_{i,j}^T$. To connect the current task T with the information from the previous task ν, the importance term Ω_{ij} is calculated as follows:

$$\Omega_{ij}^T = \sum_{\nu<T} \frac{\omega_{ij}^\nu}{(\Delta_{ij}^\nu)^2 + \xi} \tag{2.16}$$

The importance term Ω_{ij} contains two significant parts: 1) the path integral $\omega_{i,j}$ contributes the importance of each parameter to decide its change amplitude; 2) the parameters change distance $\Delta_{ij}^\nu \equiv \theta_{ij}(\nu) - \theta_{ij}(\nu - 1)$ in terms of how far the parameter moves (To avoid $\Delta_{ij}^\nu = 0$ in the denominator, a hyperparameter ξ is introduced) [27]. The definition of the equation is reasonable since the path integral Ω_{ij} only determines the contribution of one parameter to the loss function and neglects the update range of the parameter itself. If one parameter has a high Ω_{ij} but low Δ_{ij}^ν, which means this parameter is too critical and necessary to update. On the other hand, adding Δ_{ij}^ν is a normalization process of Ω_{ij} that puts the Ω_{ij} and loss function into the same order of magnitude. The introduction of the importance term limits the update rate of parameters when learning a new task. This limitation avoids the drastic changes to weights[27].

The loss function of SI can be also written in a conventional way in all regularization methods as Eq.2.2:

$$\underbrace{\mathcal{L}_{SI}(D_T)}_{\text{Total loss}} = \underbrace{\mathcal{L}(D_T)}_{\text{loss on Task T}} \underbrace{- \lambda \sum_{i,j} \cdot \Omega_{ij}^T \cdot \overbrace{(\theta_{ij} + \theta_{ij}^*)^2}^{\text{Parameters update}}}_{\text{regularization term}} \tag{2.17}$$

Memory Aware Synapses (MAS) provides another approach to continual learning by emulating synaptic behaviors, thereby endowing neural networks with a persistent memory mechanism [26]. Specifically, MAS determines the allowable update range for each parameter by computing its importance weights, which measure the sensitivity of a learned function to changes in individual parameters [26]. As illustrated in Fig.2.9(a), the method initially identifies parameter importance by contrasting the network's output (in light blue) with the ground truth labels (in green) within the training data (in yellow). Upon convergence, as depicted in Fig.2.9(b), MAS gauges the significance of parameters based on the volatility of

the output to their modifications. Within the MAS paradigm, parameter updates are contingent on their ascertained significance. Significant alterations to vital parameters are deterred, ensuring these pivotal parameters retain stability when new data is introduced. This approach ensures the model's consistent performance on antecedent tasks by maintaining their low-loss regions. Concurrently, less crucial parameters are adjusted to improve performance on new challenges, as demonstrated in task (c). This strategy ensures the model's adaptability while preserving its expertise in preceding tasks.

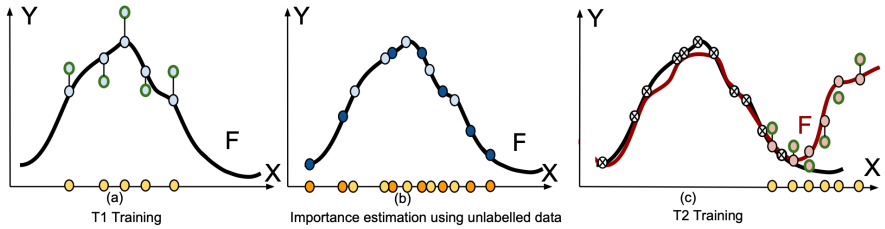


Figure 2.9: Estimate the importance of each parameter to limit the parameter change. The update of important parameters is penalized to ensure the performance of the model in old tasks.[26]

In this approach, every epoch of the training process can be considered to differentiate the parameter change path. For a given data set $D = \{X_n, Y_n\}_{n=1}^{N}$, $F(X_n; \theta)$ as the output of the neural network, and δ is a differentiation for parameter updates ($\delta = \{\delta_{ij} = \Delta\theta_{ij}\}$). The training process of the neural network can be described as follows:

$$F(X_n; \theta + \delta) - F(X_n; \theta) \approx \sum_{i,j} g_{ij}(X_n)\delta_{ij} \tag{2.18}$$

In this equation, $g_{ij} = \frac{\partial F(X_n;\theta)}{\partial \theta_{ij}}$ is the gradient of the neural network output with respect to the parameters during the training process. In the case of the multi-dimensional network output $F(X_n; \theta)$, the back-propagation algorithm is computationally difficult. To simplify the calculation, MAS suggests computing the gradients of the squared l_2 norm of the output $F(X_n; \theta)$ with respect to the parameter θ_{ij}.

$$g_{ij} = \frac{\partial \left[l_2^2(F(X_n; \theta)) \right]}{\partial \theta_{ij}} \tag{2.19}$$

During the learning periods, the importance (Ω_{ij}) of each parameter (δ_{ij}) can be described as:

$$\Omega_{ij} = \frac{1}{N} \sum_{n=1}^{N} ||g_{ij}(X_n)|| \tag{2.20}$$

By introducing the Ω_{ij} importance term, the importance of the parameters can be estimated. The update value of parameters that are important for the output in previous tasks is smaller than those that are not very necessary. This method grants that all parameters will not be uniformly changed on a large scale, only a part of the parameters that have less attribute to the outputs will be optimized for the new task, which ensures the integrity of the distribution of the parameters on old tasks. The expression of the MAS loss is as follows:

$$\underbrace{\mathcal{L}_{MAS}(D_T)}_{\text{Total loss}} = \underbrace{\mathcal{L}(D_T)}_{\text{loss on Task } T} \underbrace{- \lambda \sum_{i,j} \Omega_{ij} \cdot \overbrace{(\theta_{ij} - \theta_{ij}^{1:T-1})^2}^{\text{Params update}}}_{\text{regularization term}} \tag{2.21}$$

The MAS loss function still follows the form of the previous regularization loss formula (in Eq.2.2). The importance term Ω_{ij} determines the updated value of each parameter, which will be updated after training on a new task. To store previous information of old tasks, Ω_{ij} is the sum of all previously computed Ω_{ij}. The calculation process of Ω_{ij} does not need to go through the loss function, the available data of the old tasks can gain it. The update of Ω_{ij} is available after learning each task, which releases the computation capacity for online learning. Fig.2.10 compares different approaches' learning performance and storage space requirement. In Fig.2.10(a), the average learning performance of MAS is higher than SI, LwF, and EWC in the ImageNet with 8 tasks. Fig.2.10(b) suggests that the storage space requirement of MAS and EWC are lower than SI and LwF. The results on ImageNet show that the MAS has both higher memory performance and insufficient storage space, which is one of the ideal implementations for updating the shared parameters in the architectural methods.

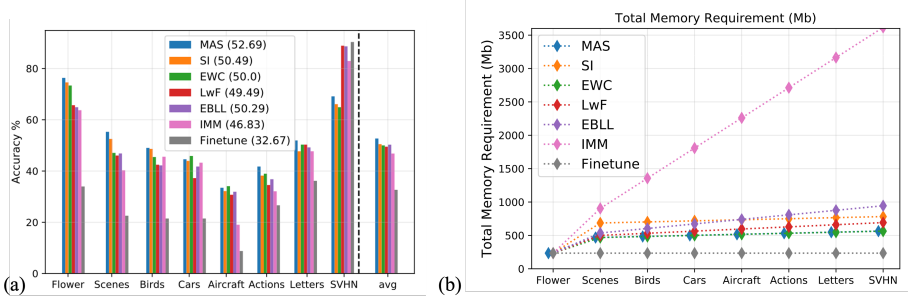


Figure 2.10: The learning performance and storage space requirement comparing between different implement[26]

2.6. Knowledge Gaps

Similar to the human brain, the architectural methods create task-specific subnetworks (regions) that often are fixed after the corresponding task is learned. How-

ever, this leads to the *network saturation* – the situation when a model does not have enough free connections to learn a new task [20]. Therefore, we can formulate the first knowledge gap for the current research:

- **Knowledge Gap 1:** Architectural methods suffer from network saturation if a large number of tasks are learned. There is no proposed solution that would solve this issue.

To eliminate this problem, we hypothesize using the regularization-based method to the least important overlapping connections in the subnetworks and allow them to change with some penalty. However, there is no example of such an application of the regularization technique to the subnetworks. Thus, the second knowledge gap is:

- **Knowledge Gap 2:** The appropriateness of regularization-based continual learning algorithms to update only part of the network and resolve the network saturation issue.

In the domain of continual learning, a common strategy to combat neural network saturation is to introduce additional connections, thereby expanding the network's capacity. Depending solely on the current network parameters to preserve information from past tasks leads to limited model reusability and necessitates additional storage space.

- **Knowledge Gap 3:** Information from previous tasks is stored within a subnetwork, enhancing parameter reusability without the need for additional storage space.

Hence, the aim of the research is to obtain a clear understanding of the formulated Knowledge Gaps. In this work, Continual Prune-and-Select (CP&S) is chosen as the representation of the architectural methods for the implementation. To overcome network saturation we examine the most basic and well-known regularization algorithms such as EWC [34], SI [27], and MAS [26]. Meanwhile, the L_2 norm is selected as the benchmark of the regularization methods.

2.7. Implementations for Dynamic Parameters Architectural methods

2.7.1. Implementation of Continual Prune-and-Select

Continual prune-and-select(CP&S)[20] is based on training a subnetwork for each given task and selecting the correct subnetwork to infer new data. The process begins with training a regular neural network for a specific task, which is then iteratively pruned to find a subnetwork that exhibits strong performance. This trained subnetwork is specifically tailored to execute the assigned task, thereby keeping the remaining network available for future tasks. When a new task emerges, a new subnetwork is determined by iteratively pruning the entire original network,

including all existing subnetworks. This is facilitated by freezing the parameters of prior subnetworks, updating the remaining parameters, and performing pruning iterations until the new subnetwork is identified. This approach enables seamless knowledge transfer between tasks without impacting performance on earlier tasks.

Each subnetwork, denoted as $\mathcal{N}^t$, is discovered utilizing the NNrelief pruning algorithm. This algorithm estimates the importance of the connection by evaluating their contributions to the total signal received by neurons. At the beginning of training on a new task, the model is firstly pre-trained on the given data. The Importance Score (SI) is calculated from the pre-trained model to measure how much each connection contributes to the receiving neuron. The definition of importance score for each neuron is defined as the ratio of the neuron output to the layer output for each input signal $\mathbf{x} = \{ x_1, x_2, \dots, x_n \}$.

$$\text{For weights: } s_{ij}^{(l)} = \frac{\left| w_{ij}^{(l)} x_i^{(l-1)} \right|}{\sum_{k=1}^{m_{l-1}} \left(\frac{1}{N} \Sigma_{n=1}^{N} \left| w_{kj}^{(l)} x_{nk}^{(l-1)} \right| \right) + \left| b_j^{(l)} \right|}$$
$$\text{For bias: } s_{bias,j}^{(l)} = \frac{\left| b_j^{(l)} \right|}{\sum_{k=1}^{m_{l-1}} \left(\left| w_{kj}^{(l)} x_n^{(l-1)} \right| \right) + \left| b_j^{(l)} \right|} \tag{2.22}$$

where $\overline{|w_{ij}x_i|} = \frac{1}{N} \sum_{n=1}^{N} |w_{ij}x_{ni}|$. NNrelief prunes connections entering neurons with low contributions to the importance score, where the sum is less than $(1 - \alpha) \sum_{i=1}^{m} s_{ij}$. Here, α serves as a hyperparameter influencing the extent of connection removal during subnetwork creation, with higher values resulting in more aggressive pruning.

In the context of task-IL, in which task ID is given in the prediction phase, the data sets arrive sequentially: $\mathbf{X}^1, \mathbf{X}^2, \dots, \mathbf{X}^T$. Pruning creates masks $\mathbf{M}^1, \mathbf{M}^2, \dots, \mathbf{M}^T$ for each task $t = 1, 2, \dots, T$, along with corresponding importance scores $S^1, S^2, \dots, S^T, S^t = \left(s_{ij}^t \right)_{i,j}$.The masks $\mathbf{M}^1, \mathbf{M}^2, \dots, \mathbf{M}^T$ are created during the pruning process for each task $t = 1, 2, \dots, T$. A mask $\mathbf{M}^t$ is represented as a matrix of binary values m_{ij}^t, where each element indicates the presence or absence of an active connection between neurons i and j in the subnetwork for task t. Specifically,

$$m_{ij}^t = \begin{cases} 1, & \text{if there is an active connection} \\ & \text{between neurons } i \text{ and } j \\ 0, & \text{otherwise} \end{cases}$$

These masks are used for constructing subnetworks that are specialized for individual tasks. The masks determine which connections are retained, enabling task-specific knowledge to be preserved while adapting to new tasks.

During the transition between different subnetworks, the shared connection parameters remain unchanged. This parameter freezing is achieved through the parameter rewriting mechanism. Once the new subnetwork is created, the task-specific segment within the subnetwork is determined. This segment encompasses

the parameters unique to the current task and not shared with any previous subnetworks. At the end of each training epoch, the model employs parameters from prior tasks to rewrite the parameters outside the current task-specific segment. By stabilizing parameters related to previous tasks, the integrity of their performance is preserved. The schematic graph of the implementation of CP&S is represented in Fig.2.11.

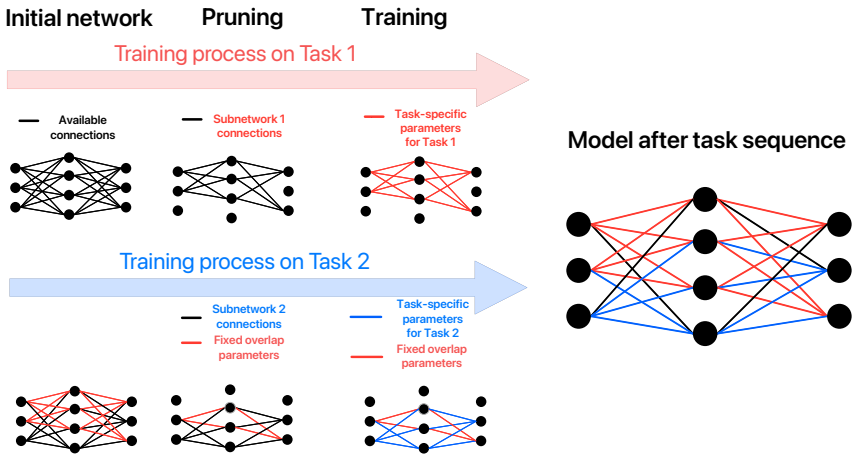


Figure 2.11: The schematic graph for the Continual Prune-and-Select (CP&S).

2.7.2. Implementation for Dynamic Parameters Architectural methods (DPA)

The Dynamic Parameters Architectural (DPA) method is designed to facilitate the adaptation of fixed parameters between subnetworks within the context of CP&S. These subnetworks are tailored using the NNrelief pruning algorithm in CP&S. However, there is a difference in the handling of parameters within overlapping subnetworks between CP&S and DPA. In CP&S, at the end of the training, only the task-specific parameters are updated for the new task, while the rest are fixed with previously stored values. Unlike CP&S, DPA allows the new subnetwork to update all parameters and not only those corresponding to task-specific areas.

On the other hand, updating the parameters in the new subnetwork alters the initial parameter distribution for the old tasks, which can result in catastrophic forgetting [2]. To prevent significant changes in the parameters within the overlapping regions, the updating of the overlap parameters is governed by the regularization loss function. Given that the neural network optimizer is gradient-based (such as

Adam or SGD), the gradient for each training epoch is determined as:

$$g_{ij} = \begin{cases} \text{gradient of the loss on current task,} & \text{For task-specific parameters} \\ \text{gradient of the loss on current task + gradient of regression term,} & \text{For overlap parameters} \end{cases} \tag{2.23}$$

Specifically, for regression problems, the base loss function $\mathcal{L}_{\text{base}}$ is the Mean Square Error, while for classification problems, it is the cross-entropy loss. The gradient $\nabla\mathcal{L}_{\text{base}}$ is computed with respect to the model output and the ground truth labels. The gradient for task-specific parameters $\mathbf{w}_{\text{task-spec}}$ is directly influenced by the task-specific loss $\mathcal{L}_{\text{task-spec}}$. In contrast, the gradient $\nabla g_{\text{overlap}}$ for the overlapping parameters $\mathbf{w}_{\text{overlap}}$ is affected by the regularization loss $\mathcal{L}_{\text{reg}}$. These independently assigned gradients effectively delineate distinct regions for parameter updates. Gradients outside the subnetwork are set to zero, ensuring that the remaining parameters remain unchanged. The schematic graph are represented in the Fig.2.12.

2.7.3. Employing regularization loss in the overlap subnetwork

In the research, four different regularization losses are employed in the overlap subnetwork, Elastic Weights Consolidation (EWC) [34], Memory Aware Snapses (MAS) [26], Synaptic Intelligence (SI) [27] and REG (L_2 norm of the parameter change between tasks) [58]. The general form of the regularization loss is represented in Eq.2.2

Elastic Weight Consolidation (EWC)

The core idea of Elastic Weight Consolidation (EWC) is to address the forgetting problem in continual learning[34]. As a method, EWC aims to protect the knowledge of previous tasks when a neural network learns new tasks. This is achieved by constraining the changes to the network's weights.

EWC adds a regularization term to the loss function, penalizing the difference between the weights and their initial values on previous tasks. By doing so, EWC encourages the network to maintain the weights learned for the old tasks when learning new ones, thereby mitigating the problem of forgetting. The specific weights to be regularized by EWC are chosen based on the Fisher information matrix, which measures how sensitive the loss function is to weight changes. Weights that are important for previous tasks are regularized, allowing them to be preserved during learning new tasks.

The loss function in Elastic Weight Consolidation (EWC) is formulated as Eq. 2.13. The process of computing the EWC loss is as follows: After training the model for the current task, the loss for the current dataset is calculated and stored in the gradient buffer. Subsequently, the Fisher information matrix is computed

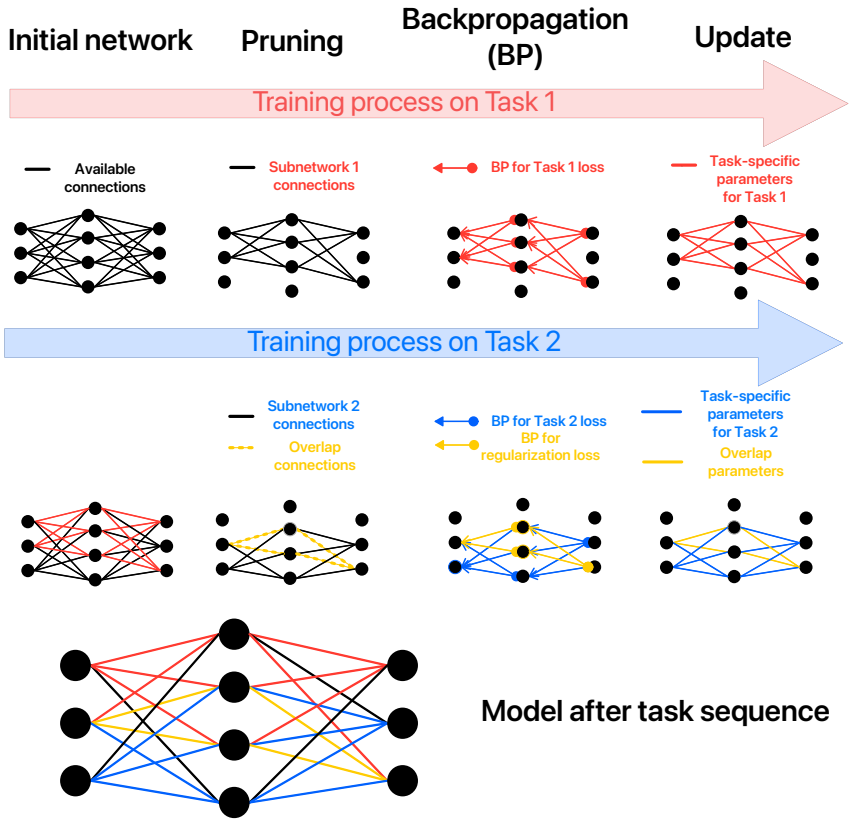


Figure 2.12: The schematic graph for the Dynamic Parameters Architectural method.

by squaring the gradients in the buffer. The size of the Fisher information matrix matches the size of the entire network. To ensure that only relevant information from the current task is retained, the Fisher information matrix is pruned using the union masks of the current subnetwork. The union mask is created by combining all task masks during the training sequence, as shown in Eq. 2.24.

$$\mathbf{M}^u = \mathbf{M}^1 \cup \mathbf{M}^2 \cup \ldots \cup \mathbf{M}^T \tag{2.24}$$

The Fisher information matrix from prior tasks is stored for future use. While learning a new task, the EWC loss focuses on the overlap areas of subnetworks.

In each training epoch, the algorithm follows a specific sequence of steps. It starts by initializing a Gradient Buffer, which helps keep track of gradients. Then, it calculates the base loss and its gradient and stores them for later use. Next, the algorithm computes the EWC loss gradient, which is then backpropagated through the model's neurons. During this process, it isolates the gradients specific to the parameters that overlap between tasks, while resetting other gradients outside this overlap area. Using the previously stored base loss gradient, the algorithm calcu-

lates gradients for both task-specific and overlapping parameters. These gradients are then used to update the network's parameters. An important aspect of this process is that the algorithm preserves knowledge of previous tasks by rewriting parameters beyond subnets using a buffer. This procedure significantly improves the model's ability to adapt and retain insights from past tasks. The training process is represented in Algorithm.2.

Algorithm 2 The training for Dynamic Parameters Architectural methods (DPA) Procedure

Initialize network
for each task t **do**
    Define overlap subnetworks O_t and the union subnetwork U_t
    for each pretrain epoch e_p **do**
        Calculate loss L_{task}
        Backpropagate loss for task-specific parameters: $\frac{\partial L_{task}}{\partial \theta_{task}}$
        Calculate regularization loss L_{reg}
        Backpropagate regularization loss for overlap parameters: $\frac{\partial L_{reg}}{\partial \theta_{overlap}}$
        network.step()
    end for
    Pruning based on network pruning algorithm[7]
    Define overlap subnetworks O_t and the union subnetwork U_t
    for each retrain epoch e_r **do**
        Calculate loss L_{task}
        Backpropagate loss for task-specific parameters: $\frac{\partial L_{task}}{\partial \theta_{task}}$
        Calculate regularization loss L_{reg}
        Backpropagate regularization loss for overlap parameters: $\frac{\partial L_{reg}}{\partial \theta_{overlap}}$
        network.step()
    end for
end for

Memory Aware Synapses (MAS)

The Memory Aware Synapses (MAS) method utilizes the output from previous task data to estimate the significance of each network parameter [26]. Similarly to the process in EWC, once training on each task is completed, the model computes the output of the model on the task-specific data. Subsequently, it calculates the L_2 norm of the output tensor and uses the resulting scalar value after normalization to derive the parameters necessary for obtaining the gradient matrix of the model, denoted g_{ij}. The importance term Ω is obtained by 2.20.

In the network, the updating of overlap parameters is governed by the gradient of the loss in MAS, while the updating of task-specific parameters is driven by the gradient of the basic loss (cross-entropy for classification or mean squared error

for regression problems). The parameters are updated by the gradients-based optimizer.

Synaptic Intelligence (SI)

Synaptic Intelligence (SI) offers a solution to combat the issue of catastrophic forgetting in neural networks [27]. It mainly revolves around the evaluation of the importance of individual synaptic connections within the network, considering their impact on overall performance. By assigning higher priority to these critical synapses during subsequent learning, the network is encouraged to make more subtle adjustments, reducing the risk of detrimental alterations that could erase knowledge from previous tasks. SI functions as a regularization technique, providing an effective approach to mitigate catastrophic forgetting. Unlike EWC and MAS, SI dynamically updates the importance term online during the training process. After each training epoch, it calculates an online parameter specification contribution term, denoted as ω_{ij}^{ν}, for each parameter θ_{ij} using Equation 2.15. The computation of ω_{ij}^{ν} relies on the basic gradients (derived from cross-entropy for classification or mean squared error for regression loss) of the parameter, scaled by the parameter's value change, which represents the difference between the updated parameter and its previous epoch value. When training is completed on the current task, the importance term Ω_{ij} for each parameter θ_{ij} is determined using Equation 2.16. The adjustment of overlap parameters hinges on the loss gradient within SI. On the other hand, task-specific parameters are updated based on the gradient of the fundamental loss, which might be cross-entropy for classification or mean squared error for regression tasks. These parameters are refined using a gradient-driven optimizer.

3

Experiments in Classification Continual learning Scenario

In this study, multilayer perceptrons (MLP) serve as the network architecture applied to various widely used MNIST-like image datasets, including MNIST [59], Fashion-MNIST [60], and EMNIST [61]. For the CIFAR-100[44], ResNET-18 [62] is selected as the network architecture

By comparing, analyzing, and summarizing the experimental data and results, the effectiveness and superiority of the work presented in this paper are substantiated.

3.1. Benchmarks

MNIST Dataset

The MNIST dataset is a widely used benchmark dataset in machine learning and computer vision. It consists of a collection of grayscale images of handwritten digits, each measuring 28x28 pixels. There are a total of 60,000 training images and 10,000 test images in MNIST, with each image corresponding to a single digit from 0 to 9. MNIST serves as a fundamental dataset for tasks such as digit classification, and its simplicity and accessibility make it a popular choice for evaluating various machine learning algorithms. The example plot for the MNIST dataset is represented in fig.3.1

Fashion-MNIST Dataset

Fashion-MNIST is a dataset designed to serve as a direct alternative to MNIST, offering similar characteristics but featuring fashion-related items instead of digits. It comprises grayscale images of 10 different fashion items, including clothing and accessories. Like MNIST, Fashion-MNIST contains 60,000 training images and 10,000 test images, all of which are 28x28 pixels in size. Fashion-MNIST provides a diverse set of visual recognition challenges and is commonly used to benchmark algorithms

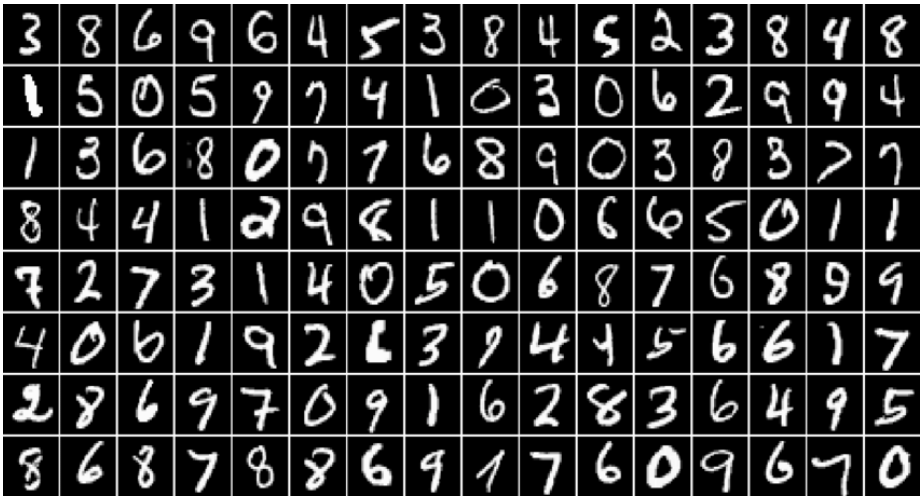


Figure 3.1: The MNIST dataset

for image classification and feature extraction in the context of fashion items. The example plot for the Fashion-MNIST dataset is represented in fig.3.2

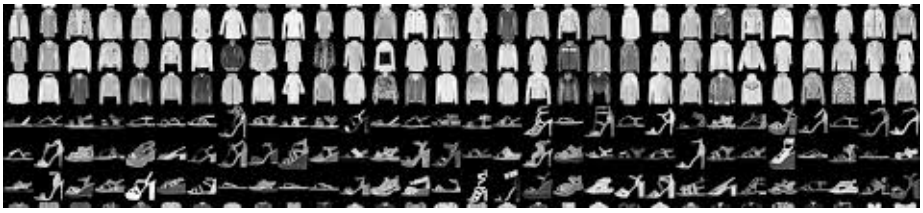


Figure 3.2: The Fashion-MNIST dataset

EMNIST Dataset

The Extended MNIST (EMNIST) dataset is an extension of the original MNIST dataset, offering a broader range of handwritten characters and symbols. EMNIST includes both digits and uppercase and lowercase letters, making it a valuable resource for tasks involving character recognition and classification. Similar to MNIST, EMNIST contains 814,255 training characters and 81,426 test characters. The dataset encompasses various writing styles and variations, providing a more comprehensive assessment of models' capabilities in handling diverse handwritten characters.

In this experiment, EMNIST Balanced is selected as the dataset for Extended MNIST. Meanwhile, to construct a regular task-sequence the data with labels 45 and 46 are removed. The example plot for the EMNIST dataset is represented in fig.3.3

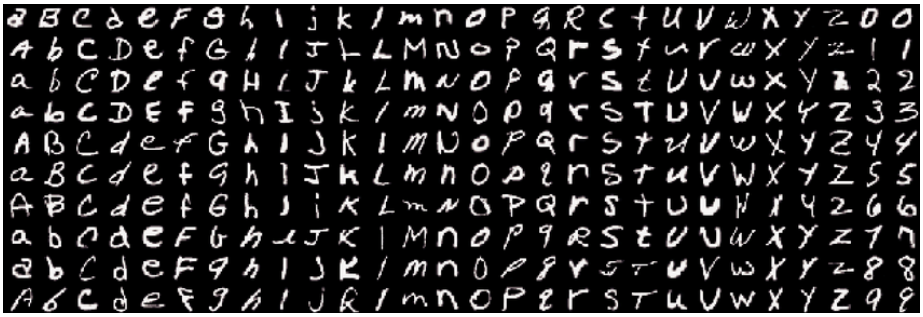


Figure 3.3: The EMNIST dataset

CIFAR-100 Dataset

The CIFAR-100 dataset is designed to provide a more diverse and challenging set of image classification tasks. CIFAR-100 consists of 100 different classes, making it a valuable resource for tasks involving fine-grained image recognition and classification. It includes 60,000 images divided into 50,000 training samples and 10,000 test samples. Each class in CIFAR-100 comprises 600 images, showcasing a wide variety of objects, animals, and scenes, which challenges models to recognize and classify images with a high degree of diversity and complexity. The dataset is an excellent benchmark for evaluating the robustness and versatility of image classification models across a rich and varied set of visual categories.

3.2. Metrics

The forgetting rate is an important indicator for measuring the learning performance of the model in the continual learning scenario. A lower forgetting rate represents the model has better performance resistance to catastrophic forgetting.

The definition of the forgetting rate is as follows:

$$\text{Forgetting Rate} = -\frac{\text{Initial error} - \text{Later error}}{\text{Initial error}} \tag{3.1}$$

where Initial error is the error of the model right after training on the initial task. And Later error is the error when revisiting that task after training on other tasks.

3.3. Task-specific parameters ratio

Task-specific parameters are defined as the parameters that only belong to one certain subnetwork. These parameters are not shared between subnetworks and only contribute to the learning and prediction for the corresponding task (shown in Fig.2.12). The task-specific parameters ratio is defined as the ratio of the task-specific parameters with the size of the subnetwork.

$$\text{Task-specific parameters ratio} = \frac{\text{Task-specific parameters}}{\text{Subnetwork size}} \tag{3.2}$$

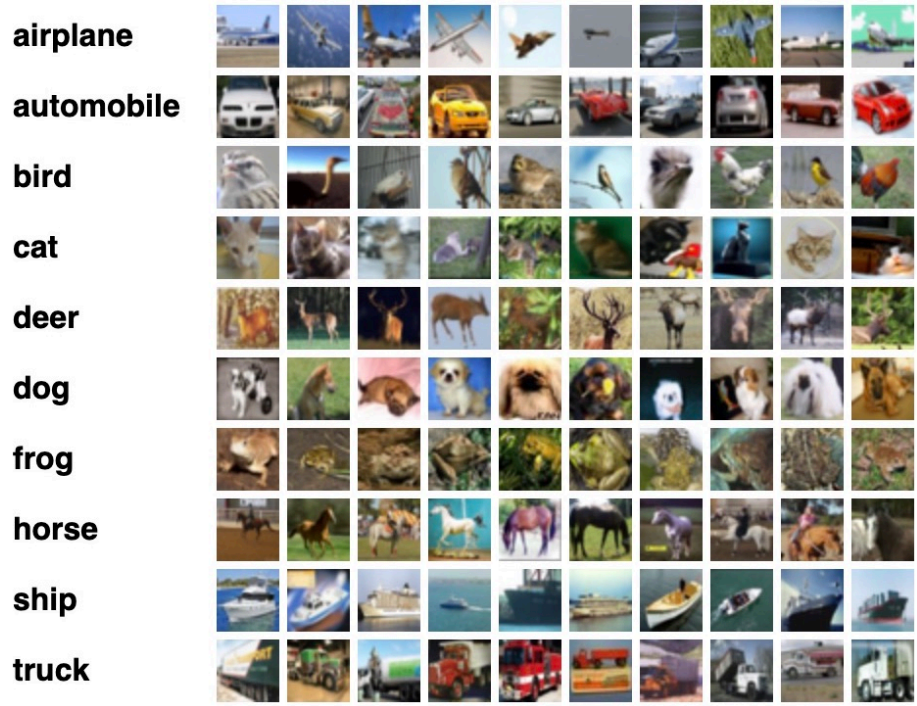


Figure 3.4: The CIFAR-100 dataset

By analyzing the ratio of task-specific parameters, the degree to which each subnetwork specializes in its task can be inferred. A higher ratio indicates that a significant portion of the subnetwork is dedicated solely to its specific task, suggesting a higher degree of task specialization. Conversely, a lower ratio may imply that the subnetwork relies more on shared or general parameters, indicating less task-specificity. This metric offers insights into how neural architectures strike a balance between task-specific features and shared representations.

3.3.1. Multi-Head Model classification continual learning scenario

The Multi-Head Model is a neural network architecture used in continual learning scenarios. This model consists of a single overarching architecture with multiple individual "heads", each corresponding to a specific task, such as Task 1, Task 2, and Task 3. Each head operates independently and is dedicated to processing the data and generating predictions for its respective task. A schematic graph illustrating the concept of multi-heads can be found in Fig.3.5.

In the context of continual learning, the Multi-Head model is designed to solve situations involving multiple distinct tasks. This approach offers several advantages:

- Distinct Task Handling: The model has separate "heads" for each task (e.g.,

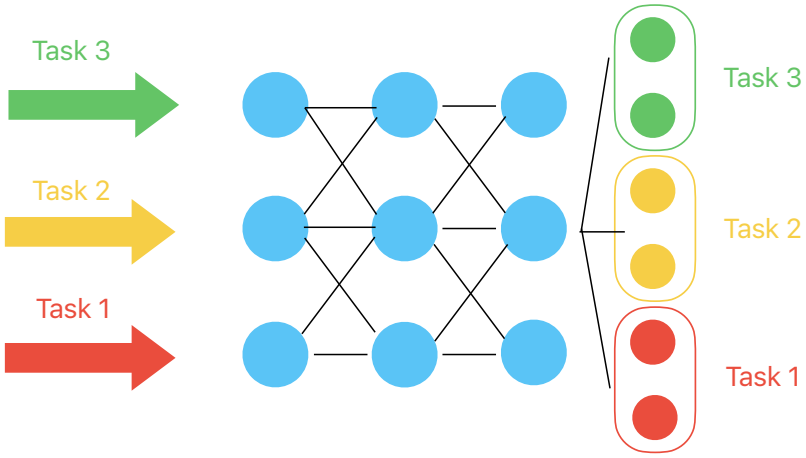


Figure 3.5: The schematic graph for multi-heads in a continual learning scenario.

Task 1, Task 2, Task 3). These heads work independently, focusing solely on their assigned task to ensure the model's output aligns with the current task's requirements.

- Task-Specific Loss Computation: The Multi-Head model computes the categorical cross-entropy loss for the digits relevant to the current task. This targeted loss calculation fine-tunes the model specifically for the ongoing task.

- Mitigation of Catastrophic Forgetting: By using dedicated heads and task-specific loss computation, the model minimizes interference between newly learned information and previously learned tasks. This approach helps prevent catastrophic forgetting, preserving task-specific knowledge.

- Scalability and Adaptability: The model is highly flexible and can easily accommodate new tasks by adding additional heads. This design allows the model to expand its capabilities without requiring extensive changes to its architecture.

In summary, the Multi-Head model provides a structured yet adaptable solution to the challenges of continual learning, ensuring both specificity and flexibility across multiple tasks. In this research, the Multi-Head model is selected for the model in classification continual learning scenarios. The head of different tasks is not shared, and the heads are updated by the basic loss function. The regularization loss is not applied in the heads.

3.4. Experiments

3.4.1. MNIST dataset

To construct the task sequence, the MNIST dataset is divided into 5 tasks with 2 classes in each task. Each task's dataset D_t consists of MNIST handwritten digit images with class labels. These images are created from pixel data that has undergone a consistent random arrangement at different time intervals during various task phases. In this chapter, a comparison is made among four regularization methods, namely EWC, MAS, SI, and L_2, when applied to shared parameters in Continual Prune-and-Select (CP&S). The aim is to release the constrained differences and enhance the model's learning capability for new tasks. For the neural network architecture, a Multilayer perceptron (MLP) with 2 hidden layers and 400 neurons per layer is applied. Between each hidden layer, set ReLU as the activation function. The training batch size for each training is 64. The hyperparameter λ for each regularization method (in Eq.2.2) is selected by grid-search, which is represented as follows:

Model name	λ
DPA-EWC	4500
DPA-MAS	15
DPA-REG	3
DPA-SI	3

Table 3.1: λ value for different models for MNIST dataset

Fig. B.1 illustrates the learning capacity of various models when applied to the MNIST dataset. Fig. B.1a presents the error of different models in predicting tasks as the task sequence progresses. Within each gray region, the curves represent changes in error for the current task after training on that task. For instance, in the first gray region, each curve comprises 5 points, with each point representing the recognition error of the model for Task 1. The first point represents the recognition error when the model is trained solely on Task 1, while the second point reflects the recognition error for Task 1 after the model has been trained on Task 2. By studying the downward trend of the recognition error curve for each task, the neural network's anti-forgetting capability can be demonstrated.

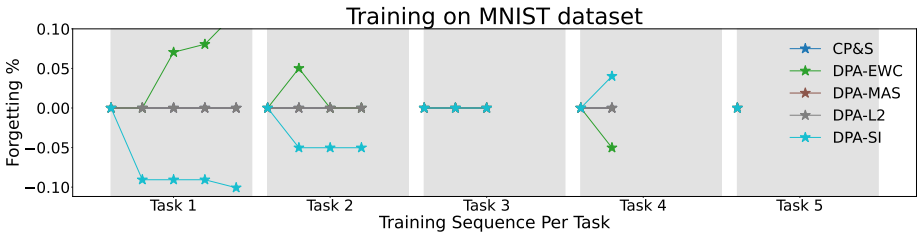
From Fig. 3.6a, all models achieve error levels less than 1% on the MNIST dataset. The slight downward trend in each model's respective curve signifies their better anti-forgetting performance on the MNIST dataset, indicating their effectiveness in preserving a substantial amount of information even after learning new tasks. Notably, CP&S, DPA-MAS, and DPA-L_2 exhibit consistently unchanging error rates for each task, suggesting no forgetting of information from previous tasks. However, DPA-EWC shows a significant drop in recognition error for old tasks, indicating a higher propensity for forgetting old task information. In contrast, DPA-SI exhibits a notable behavior termed as *positive transfer* [63], where the performance on previous tasks improves after learning new ones. This effect suggests that the acquisition of new tasks might enhance the model's understanding and representa-

tion of earlier tasks, possibly due to shared features or hierarchical representations.

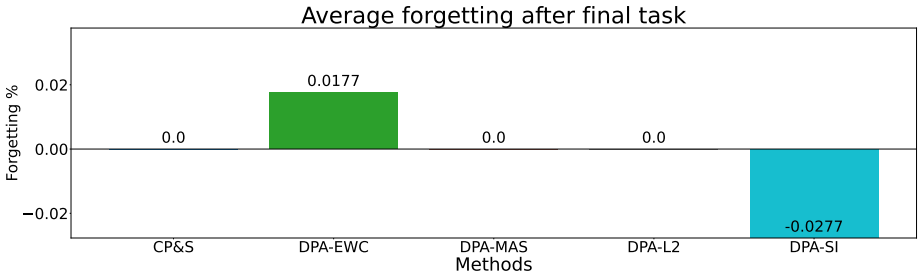
In Fig.B.1c, the performance metrics of different continual learning methodologies are presented over a series of tasks. The incorporation of network pruning in both DPA and CP&S methods endows them with a heightened resilience against catastrophic forgetting, especially when compared with pure regularization strategies. Additionally, their proficiency in learning new tasks markedly exceeds that of the regularization methods. Within the depicted methodologies, our custom-designed DPA-based techniques stand out, surpassing both the CP&S approach and traditional regularization strategies. While CP&S is designed to inherently prevent forgetting, the DPA techniques excel further by not just resisting forgetting, but also by displaying higher adaptability and learning prowess on new tasks—this is evident from their consistently minimized error rates.

Fig. 3.6 illustrates the forgetting rates as the task sequence progresses in the MNIST dataset. In Fig. 3.6a, we can observe that these forgetting rates provide insights into the models' ability to retain previously learned information when exposed to new tasks. In this plot, several models' forgetting rates are notable. CP&S exhibits a forgetting rate of 0, indicating that it does not forget information from prior tasks and maintains a consistent performance throughout the task sequence. This is because the parameters from the previous task are fixed. DPA-EWC, on the other hand, demonstrates a slight forgetting rate of 0.0177, implying that it retains most of the previously learned knowledge but experiences a marginal decrease in error when learning new tasks. DPA-MAS and DPA-L_2 both demonstrate a forgetting rate of 0, signifying their effective retention of information from previous tasks even in situations where parameters from the previous model have been modified. When we combine this observation with the insights provided by Fig. B.1, it becomes evident that the dynamic parameter architecture method exhibits higher performance in learning new tasks while also maintaining a memory of previously learned information, avoiding forgetting. Remarkably, DPA-SI displays a negative forgetting rate of -0.0277, indicating a positive transfer of knowledge. This means that learning new tasks somehow improves the model's performance on old tasks, enhancing its ability to recognize previous patterns.

In Fig. 3.7, we observe the average recognition error across the task sequence. Notably, the average error for all models remains consistent and doesn't decrease as the task sequence progresses. This suggests that the models exhibit resistance to catastrophic forgetting. Meanwhile, DPA-L_2 consistently outperforms other models throughout the entire task sequence. This indicates that the DPA-L_2 method has higher performance in both learning new tasks and retaining previously learned information, making it a standout performer. Moreover, an interesting trend becomes evident: Once the third task is learned, all DPA methods outperform CP&S in terms of average error. Specifically, DPA-EWC maintains a lower average error even when there are positive forgetting rates for the fourth and fifth task sequences. This illustrates that DPA methods, by releasing fixed parameters, substantially improve the model's capacity to grasp new tasks. Even when forgetting happens, they consistently do better than CP&S. In summary, DPA methods offer a distinctive trade-off between learning and forgetting, highlighting their adaptability in maintaining a



(a) Average forgetting rate for all observation tasks sequence



(b) Average forgetting rate after learning the final task

Figure 3.6: The forgetting rate in the MNIST data experiment

balance between these two pivotal aspects of continual learning.

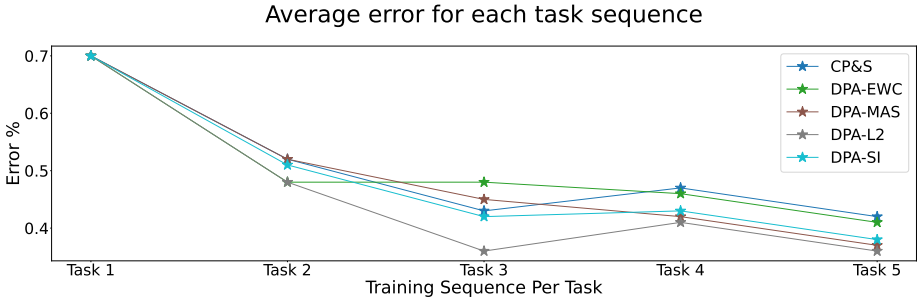


Figure 3.7: The average recognition error for all observation tasks in the MNIST data experiment

Fig. B.2 (in Appendix.B.3) shows network sparsity in the MNIST data experiment. In Fig.B.2a, we see how the proportion of task-specific parameters (defined in Eq.3.2) changes as we go through different tasks. In this plot, each row represents how many of the parameters in each subnetwork are specific to the task, and it changes as we move from one task to the next. For instance, in the initial row, when we commence with the first subnetwork, we observe that all the parameters are entirely dedicated to that particular task, resulting in a 100% task-specific parameter ratio. However, as we progress through the comprehensive training of the entire task sequence within the MNIST dataset, a significant shift occurs. This ratio steadily declines and eventually stabilizes at around 11%. This finding implies that

approximately 90% of the parameters within the subnetwork become shared with other subnetworks, leaving only 12% that remain uniquely associated with Task 1.

Additionally, our investigation reveals a consistent trend across multiple subnetworks. Most of them converge to having just 8-14% of task-specific parameters remaining after the culmination of the entire task sequence. This indicates that the majority of parameters are indeed shared among different networks. The result of this analysis is that the model efficiently uses shared parameters for different tasks. This means the model can reuse and apply a significant part of what it learned from one task to another. This sharing of parameters is crucial in continual learning because it helps the model adapt to new tasks while keeping the knowledge it gained from previous ones.

In Fig.B.2b, we look at the ratio of the union mask compared to each subnetwork. Each subnetwork takes up only 4-8% of the total model's parameters. After finishing training on the task sequence, the union mask contains around 15% of the connections in the network. This means that the model is quite sparse, with 85% of the connections unused. This sparsity arises from the fact that handwritten digits are usually centered in the images, and pixels near the edges don't provide much information for classification.

3.4.2. Fashion-MNIST dataset

The neural network architecture, batch size, pruning iteration and α_{fc} used in the Fashion-MNIST dataset experiment remain consistent with those employed in the MNIST dataset experiment. In the Fashion-MNIST dataset experiment, the hyperparameter λ for each regularization method (as defined in Eq.2.2) is determined via a grid-search approach and is presented in Table 3.2:

Model Name	λ **Value**
DPA-EWC	10000
DPA-MAS	4.5
DPA-REG	4.5
DPA-SI	2

Table 3.2: λ Values for Different Models in the Fashion-MNIST Dataset

The graph in Fig.B.3 illustrates the recognition error as the task sequence progresses in the Fashion-MNIST dataset. Due to the Fashion-MNIST dataset's higher complexity compared to MNIST, the experiment's error falls in the range of approximately 3.7% to 4%, slightly below that observed in the MNIST dataset. Similar to previous experiments, forgetting is not a prominent issue in the Fashion-MNIST dataset. Notably, DPA-EWC shows a significant error decrease during Task 1, indicating its lower resistance to forgetting compared to other models.

It's important to note that on the final task, CP&S achieves an error of 3.58%, while DPA-EWC records 3.38%, DPA-MAS reaches 3.63%, DPA-L_2 maintains 3.7%, and DPA-SI excels with an error of 3.28%. These results indicate that the dynamic parameters architecture method, particularly with the application of EWC and SI to release shared fixed parameters, exhibits better performance in learning new tasks

within the Fashion-MNIST dataset. In Fig.B.3c, we see the performance of various continual learning methods across tasks. DPA, CP&S, and regularization methods perform similarly in the initial tasks. However, by Task 3, regularization methods show a higher error rate, while DPA and CP&S remain stable. This indicates that DPA and CP&S are more better than regularization methods in the Fashion-MNIST dataset.

In terms of average recognition error, DPA-EWC exhibits the lowest performance due to its relatively high forgetting rate. Conversely, DPA-SI achieves the highest average recognition error after completing all five tasks, primarily because of its substantial positive transfer. The remaining models perform similarly to each other. This observation highlights the presence of varying degrees of positive transfer within the DPA framework during the Fashion-MNIST experiment. Positive transfer indicates that the model excels not only in learning new tasks but also in enhancing its recognition performance on previously learned tasks. Consequently, the DPA approach outperforms CP&S in terms of recognition error, even when considering the presence of forgetting. This highlights the potential advantages of incorporating positive transfer as an integral part of the continual learning process.

For the average recognition error, DPA-EWC has the worst average recognition error due to its high forgetting rate. Due to the high positive transfer, DPA-SI has the highest average recognition error after learning all five tasks. the rest of each model is close to each other. this phenomenon indicates that, in the Fashion-MNIST experiment, there are varying degrees of positive transfer in DPA. The model not only performs well on new tasks but also improves the recognition rate of old tasks. Therefore, DPA performs better than CP&S in the recognition error under the premise of allowing forgetting.

Fig.B.5 illustrates the task-specific parameters ratio and the network's sparsity following the completion of all task sequences. Subnetwork 1 and subnetwork 5 exhibit a task-specific parameters ratio ranging from 9-13%. This implies that approximately 90% of the parameters are shared among different networks for these subnetworks. In contrast, subnetworks 2, 3, and 4 have a higher task-specific parameters ratio compared to subnetworks 1 and 5. This disparity arises due to the larger sizes of subnetworks 2, 3, and 4 (shown in Fig.B.5b). However, it is important to note that the union subnetwork's size occupies only about 12.5% of the total network's parameters. This indicates that the model remains highly sparse despite the variations in task-specific parameters. The prevalence of sparsity in the model is due to the nature of handwriting digit datasets, where the majority of pixels near the image boundary contain little discriminative classification information, leading to an efficient utilization of parameters.

In the context of the Fashion-MNIST experiment, the dynamic parameter architecture (DPA) framework exhibited good performance characterized by two significant attributes. Firstly, it prominently showcased a higher degree of positive transfer, allowing the model not only to excel in learning new tasks but also to enhance its recognition accuracy on previously acquired tasks. This observation underscores the practical utility of DPA, as it facilitates knowledge retention and application across a sequence of tasks. Secondly, it is important to note that the

DPA framework showed the capability to achieve overall learning performance similar to or better than the CP&S approach. This held even in situations where the model was allowed to forget. The DPA framework provides a trade-off between forgetting and learning new tasks. Lastly, it is essential to highlight that the model maintained a high level of sparsity despite its better performance in the Fashion-MNIST experiment. New subnetworks tend to be generated on already trained subnetworks.

3.4.3. EMNIST dataset

The neural network architecture, batch size, pruning iteration, and α_{fc} used in the EMNIST dataset experiment remain consistent with those employed in the MNIST dataset experiment. In the EMNIST dataset experiment, the hyperparameter λ for each regularization method (as defined in Eq.2.2) is determined through a grid-search approach and is presented in Table 3.3:

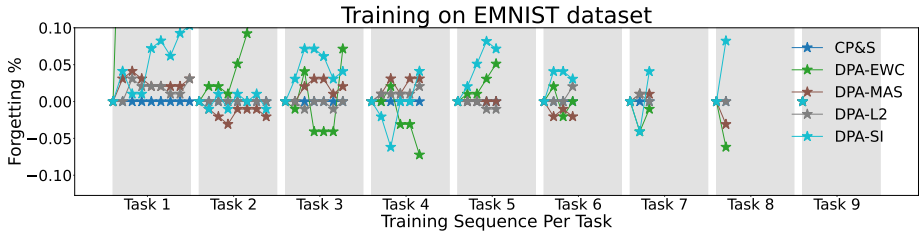
Model Name	λ **Value**
DPA-EWC	10000
DPA-MAS	4.5
DPA-REG	10
DPA-SI	2

Table 3.3: λ Values for Different Models in the EMNIST Dataset

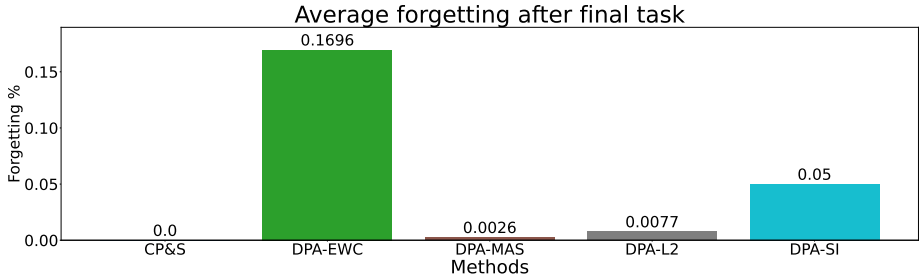
Most of the results of the experiment in the EMNIST dataset are in the Appendix.B.5. In contrast to the MNIST and Fashion-MNIST dataset experiments, the EMNIST dataset is divided into 9 tasks. As the number of learning tasks increases, old information in neural networks becomes more susceptible to being overwritten by new information. In Fig.B.7a, we observe that, apart from DPA-EWC, the model's recognition error on the tasks hardly decreases as the number of learned tasks grows. This observation indicates that the model possesses a high resistance to forgetting. Additionally, in Fig.B.7b, DPA-MAS, DPA-L_2, and DPA-SI demonstrate better learning capacity in the final task, with DPA-L_2 achieving the highest error for the final task. This phenomenon suggests that by freeing the shared parameters, the model can involve more parameters in learning new tasks, ultimately leading to improved performance on new tasks. In Fig.B.7c, the chart shows error rates for various continual learning methods. DPA and CP&S methods have lower error rates than the regularization methods in most tasks. As more tasks are introduced, the error rates rise, yet DPA and CP&S remain consistently stable. This indicates that DPA and CP&S possess a higher resistance to catastrophic forgetting compared to regularization techniques.

In the context of the EMNIST dataset, which involves a significant number of tasks, the challenge of retaining knowledge from previous tasks becomes increasingly evident as the model progresses through its learning sequence. This results in fluctuations in the model's recognition error when evaluating its performance on older tasks. These error fluctuations indicate that as the model acquires new knowledge, it can experience positive transfer, where its understanding of certain

concepts or patterns improves even for previously learned tasks. However, a closer examination in Fig.3.8b reveals that each DPA model exhibits a positive forgetting rate, indicating that despite the positive transfer phenomenon, these models cannot completely eliminate forgetting. The average forgetting rates for DPA-MAS and DPA-L_2 are relatively low, at 0.0026% and 0.0077%, respectively. While these rates are minimal, they demonstrate that some degree of forgetting still occurs, albeit on a negligible scale.



(a) Average forgetting rate for all observation tasks sequence



(b) Average forgetting rate after learning the final task

Figure 3.8: The forgetting rate in the EMNIST data experiment

Fig.3.9 shows the average recognition error for each model. Notably, DPA-L_2 and DPA-SI consistently outperform CP&S in terms of average error across the task sequences. Particularly, DPA-L_2 achieves the lowest average error upon completing training on all tasks. These results indicate that in complex continual learning scenarios like EMNIST, the trade-off between learning and forgetting becomes more apparent compared to simpler datasets. In such cases, DPA models exhibit a significant improvement in average recognition error when compared to CP&S. Therefore, with a larger number of tasks, DPA's performance improvement over CP&S becomes more pronounced.

In scenarios with a high number of tasks, the neural network maintains a consistently high level of sparsity, as evident in Fig.B.8. Upon completing the training, the ratio of task-specific parameters reduces to less than 10%. This reduction implies that over 90% of the parameters within each subnetwork are shared with other subnetworks, resulting in diminishing task-specific parameters. Specifically, each subnetwork comprises only 5-10% of the total neural network's parameters, while the union of subnetworks after completing the task sequence accounts for approximately 30%. This high level of sparsity indicates that new subnetworks

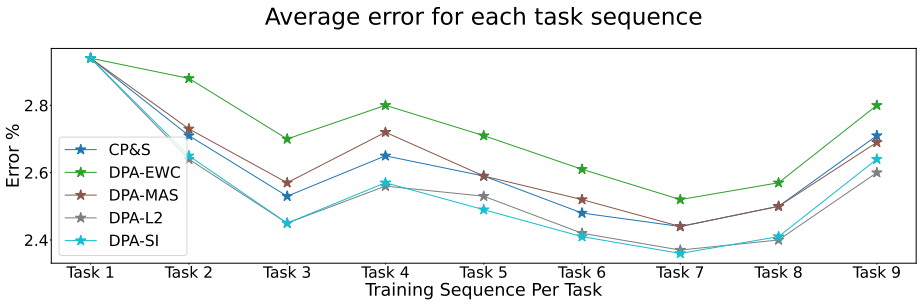


Figure 3.9: The average recognition error for all observation tasks in the EMNIST data experiment

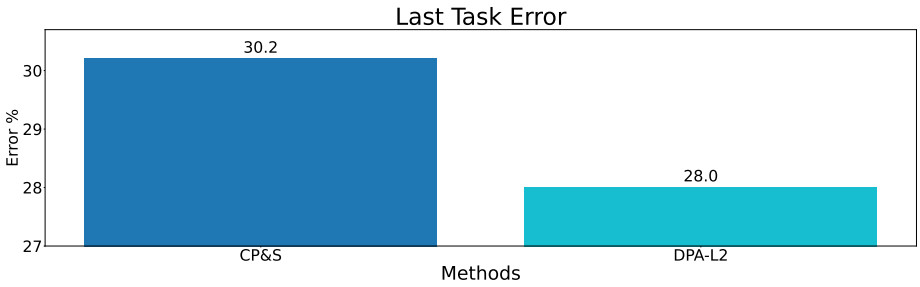
tend to emerge within existing subnetworks. Furthermore, there remain numerous connections that are left unused.

3.4.4. CIFAR-100 dataset

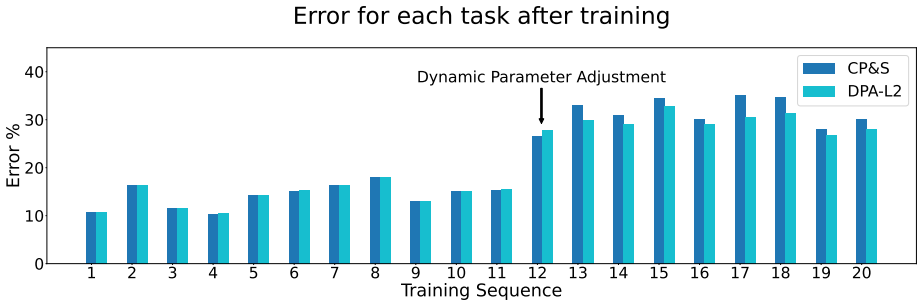
To construct the task sequence, the CIFAR-100 dataset is partitioned into 20 tasks, each consisting of 5 classes. Each task's dataset, denoted as D_t, comprises CIFAR-100 digit images with corresponding class labels. To ensure optimal learning performance, we have adopted the L_2 regularization method for updating the overlap parameters between different subnetworks, as it has been demonstrated to be effective in MNIST-like datasets. The hyperparameter λ for each regularization method (as shown in Eq. 2.2) is set to 15. In our research, inspired by the findings in CP&S [20], it was observed that the model's learning ability starts to decline and network saturation occurs when learning more than 11 tasks. Consequently, we have implemented a dynamic parameter adjustment strategy. After completing the training for task 11, the model transitions to the original CP&S implementation, where subnetworks are fixed after training. At the outset of task 12, the previously fixed parameters are released to participate in the training of the current task, facilitated by the application of L_2 regularization. Moreover, the hyperparameters used for training on CIFAR-100 remain consistent with those employed in the CP&S research [20].

Fig. 3.10 presents a comparison between DPA-L_2 and CP&S models on the CIFAR-100 dataset, which is divided into 20 distinct tasks. Within this figure, Fig. 3.10a specifically highlights the differences in performance between the two models, focusing on their results for the final task. In this comparison, DPA-L_2 shows reduced recognition errors for the final task. This observation suggests that the DPA-L_2 model's strategy, which involves releasing overlapping parameters and applying L_2 regularization, effectively increases the number of parameters participating in the learning process throughout the tasks. As a result, this approach appears to enhance the model's ability to learn and adapt to new tasks.

Furthermore, Fig. 3.10a offers a detailed perspective on the model's learning performance during the final task sequence. This analysis evaluates the model's accuracy after completing the training for the final task and making predictions for



(a) Recognition error for the last tasks sequence.



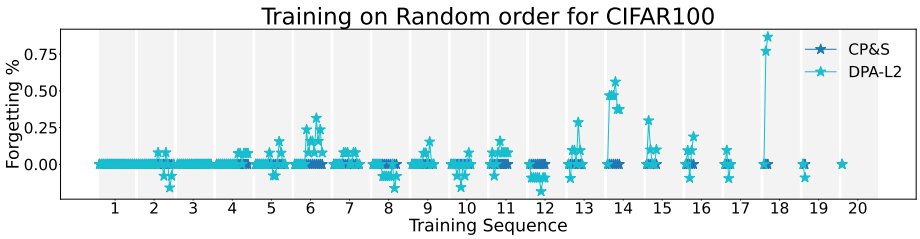
(b) Average forgetting rate for the final task sequence.

Figure 3.10: The performance comparison in the CIFAR-100 data experiment.

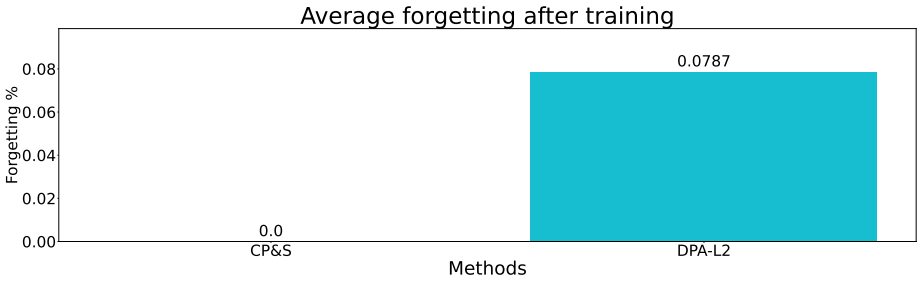
all preceding tasks. An important takeaway from this figure is the minor fluctuation in the error rate observed for task 12. However, starting from task 13 and beyond, DPA-L_2 consistently maintains a lower prediction error compared to CP&S. This trend is noteworthy because it signifies an enhancement in model performance without a significant reduction in accuracy for tasks completed prior to task 11. This aspect is crucial as it indicates that DPA-L_2 releases the network saturation issues while preserving the model's knowledge retention of previous tasks.

Fig.3.11a presents the forgetting rate observed in the training history of the task sequence. The forgetting curve of DPA-L_2 exhibits fluctuations as the task sequence progresses, indicating that, despite releasing overlap parameters, the model experiences some degree of forgetting. In contrast, the forgetting curve of CP&S remains relatively flat, suggesting no forgetting. This observation is further supported by the analysis of the average forgetting rate after completing the final task, as depicted in Fig.3.11b. Here, CP&S records a 0% forgetting rate, while DPA-L_2 demonstrates forgetting for previous tasks.

Additionally, the graph in Fig. 3.12, illustrating the average recognition error across all tasks, demonstrates an improvement following the implementation of dynamic parameter adjustment in the DPA-L_2 model after completing 12 tasks. This decrease in recognition error indicates that DPA-L_2, through its dynamic parameter adjustment strategy, relieves the issue of network saturation by releasing parameters and enhances the overall accuracy of the model's task recognition. This im-



(a) Average forgetting rate for all observation tasks sequence



(b) Average forgetting rate after learning the final task

Figure 3.11: The forgetting rate in the CIFAR-100 data experiment

provement represents a positive development when compared to the conventional CP&S approach, underscoring the effectiveness of dynamic parameter adjustment in handling complex learning scenarios. The improvement of the overall performance of the DPA-L_2 indicates that, by releasing the fixed parameters, while allowing small-scale forgetting, the model improves overall learning capabilities by improving learning performance on new tasks.

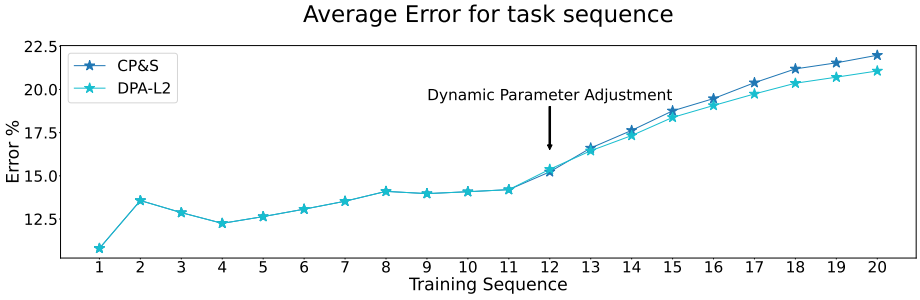


Figure 3.12: Average recognition error after the entire task sequence.

Impact of the class correlation to the learning capacity

In DPA, the subnetworks are allowed to share the parameters with each other. The shared parameters are updated based on the information both on previous tasks and current tasks. In this case, if the types of tasks before and after are similar, the

learning of the model may be affected by the similarity of the tasks. Therefore, we divided the CIFAR-100 dataset into 20 different tasks according to the classification of the superclass (each superclass includes 5 classes) and analyzed the learning ability of the model in the task sequence constructed by the superclass, especially the learning ability under different task types. The task sequence based on the division of the superclass is shown in Table.3.4.

1. insects	11. large natural outdoor scenes
2. fruit and vegetables	12. aquatic mammals
3. flowers	13. large carnivores
4. vehicles 2	14. large omnivores and herbivores
5. vehicles 1	15. household furniture
6. food containers	16. large man-made outdoor things
7. small mammals	17. household electrical devices
8. people	18. non-insect invertebrates
9. fish	19. reptiles
10. medium-sized mammals	20. trees

Table 3.4: Create task sequence based on the division of superclasses (Superclasses 1 order).

he recognition error of both CP&S and DPA-L_2 exhibits significant fluctuations after task 12, which can be attributed to network saturation. This saturation diminishes the model's capacity to learn new tasks effectively. DPA-L_2 shows more pronounced fluctuations across different task types compared to scenarios with fewer task sequences. Nonetheless, the fluctuation patterns observed in DPA-L_2 and CP&S are notably similar. This indicates that, even after releasing overlap parameters, DPA-L_2 does not demonstrate a distinctly different learning capability for various class types compared to CP&S. This suggests that, in the existing experiments, releasing overlap parameters has not significantly demonstrated knowledge transfer between task types, indicating a potential area for further investigation in enhancing the models' adaptability to diverse class categories.

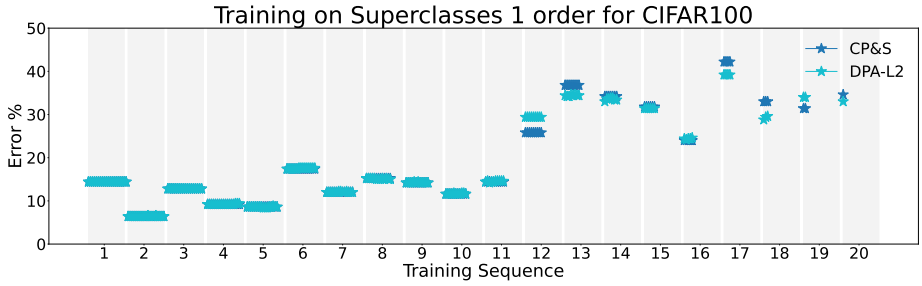


Figure 3.13: Average recognition error after the entire task sequence for the Superclasses 1 order.

3.4.5. Discussions

By analyzing the experiments in MNIST-like and CIFAR-100 datasets, the discussion of the DPA in the classification continual learning scenario is as follows:

- **Higher Learning Capabilities:** DPA consistently has a higher learning performance when faced with a longer or more complex task sequence. Its flexible approach to adjusting shared parameters enables the model to retain past knowledge while effectively learning new tasks, thereby enhancing its overall learning capacity than CP&S.
- **Trade-off Between Learning and Forgetting:** DPA balances the acquisition of new task-related knowledge and the retention of previously learned information. Under the premise of allowing some level of forgetting, DPA consistently achieves lower average recognition error. This adaptability makes the model perform well in continual learning scenarios, allowing it to outperform CP&S.
- **Higher Performance with L_2 Regularization:** DPA-L_2 achieves the highest average error across the MNIST-like datasets. Fine-tuning the reused parameters with L_2 regularization maintains low overall error[64]. This demonstrates that parameter reuse is effective for retaining information from past tasks and provides a trade-off between learning and forgetting, making L_2 regularization particularly effective among other regularization methods in DPA on MNIST-like and CIFAR-100 datasets.

4

Experiments in Regression Continual learning Scenario

4.1. Introduction

In the past few years, constitutive modeling has undergone significant transformations, primarily due to the integration of data-driven analysis framework[65]. Traditional models came with several assumptions and required intensive computations. In contrast, data-driven analysis offers improved accuracy, speedier calculations, and more adaptability. As highlighted in [66], the data-driven approach begins with the examination of diverse microstructures under predetermined conditions. Techniques such as the Finite Element Method (FEM) are applied to particular Representative Volume Elements (RVEs). Once computations are done, a comprehensive database from these RVEs is constructed. This database aids Machine Learning (ML) in identifying relationships between the structure and the outcomes. Consequently, it becomes possible to predict the overarching properties of unexplored materials. By combining classic computational techniques with modern data-driven methods, we can make highly precise predictions about material properties, merging two research domains.

In the data-driven framework, the material properties of a single RVE can be treated as an individual learning task. When the data from each RVE is received at different times, the tasks that come one after another create a sequence of tasks. Therefore, in the data-driven system, data continuity can be considered as a regression problem in continuous learning.

4.2. Benchmarks

Hyperelasticity is a material property that characterizes the behavior of certain elastic materials. These materials are often referred to as ”hyperelastic materials” or ”elastic material models.” Hyperelasticity describes the ability of these materials to undergo elastic deformation when subjected to various stress states and maintain

their elastic nature throughout. This behavior is typically modeled using mathematical formulations known as "strain energy density functions." Unlike linear elastic materials, hyperelastic materials do not conform to Hooke's law, making it more challenging to describe their mechanical behavior. In hyperelastic materials, the stress-strain relationship is nonlinear, and they exhibit a unique ability to undergo significant elastic deformation without permanent distortion, thereby defying conventional linear elasticity principles. Utilizing data-driven analysis, the mechanical properties of hyperelastic materials can be rapidly predicted with minimal computational burden. In a data-driven analysis system, the Representative Volume Element (RVE) functions as the smallest unit for assessing material properties, highlighting the unique characteristics of diverse materials. When it becomes necessary to study various materials within a single data-driven analysis system and the data pertaining to these different materials arrive sequentially, addressing the challenge of learning in sequential tasks can be viewed as a continual learning problem.

The dataset for the regression continual learning scenarios is the hyperelastic properties of the composite with parallel fiber and perpendicular fibers. The schematic plot for the dataset is represented in Fig.4.1.

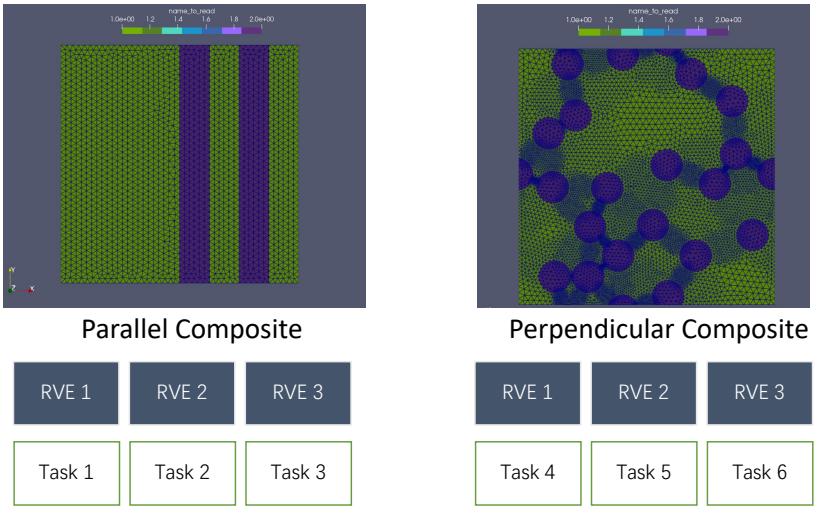


Figure 4.1: Hyperelasticity dataset

In this research, both the Parallel Composite and Perpendicular Composite datasets are categorized into three distinct tasks, each corresponding to a different composite. The training data for each task comprises in-plane stress values (σ_x, σ_y, and τ_{xy}) and strain values (ε_x, ε_y, and ε_{xy}) specific to the Composite. These stress and strain datasets are generated using Abaqus software [67]. The author of this data is O. Taylan Turan. Each task within both the Parallel and Perpendicular datasets consists of approximately one thousand stress-strain vectors. In these regression problems, the input data for each task consists of strain vectors, while the output

data represents the corresponding stress vectors. Additionally, for each dataset, the order is shuffled, and the average error is analyzed across six different task orders of the dataset.

Given the significant disparities in the hyperelastic relationships of Parallel Composite and Perpendicular Composite, neural networks exhibit distinctive learning behaviors when exposed to these dissimilar datasets. In the realm of continual learning, task sequences encompass a range of task types. To evaluate the model's capacity for continual learning across diverse task types, the model will be trained on the sequence with variant task types based on the difference in the hyperelasticity of different composites. Further, the Mix dataset is created by combining the Parallel dataset and the Perpendicular dataset. The Mix dataset comprises six tasks, with the dataset order being randomized for analysis. We assess the average error across six distinct task orderings within this dataset. To investigate the model's continuous learning capability in response to changes in task types, the first three tasks consist of Parallel datasets, while the last three tasks are from Perpendicular datasets.

4.2.1. Relative Error for the Regression Problem

In regression problems, the relative error is a crucial metric for evaluating the accuracy of predictions. It measures the discrepancy between the predicted values (y_{pred}) and the actual values (y) relative to the magnitudes of the actual values. The relative error is defined as:

$$\text{Relative Error} = \frac{\|y - y_{\text{pred}}\|_2}{\|y\|_2}, \tag{4.1}$$

where:

$$\begin{aligned} y &: \text{Actual values} \\ y_{\text{pred}} &: \text{Predicted values} \\ \|\cdot\|_2 &: L_2 \text{ norm (Euclidean norm)} \end{aligned}$$

This metric helps assess the predictive accuracy of regression models, taking into account the scale of the true values. A lower relative error indicates better model performance in approximating the target values. In practical terms, a relative error of zero would imply that the model's predictions are an exact match to the actual values. In this section, relative error is employed to estimate the learning performance of each model.

4.3. Experiments

In this section, we compare four regularization techniques: EWC, MAS, SI, and L_2. These methods are used on shared parameters in the Continual Prune-and-Select (CP&S) framework. The goal is to reduce constraints and improve the model's ability to adapt to new tasks. For the neural network architecture, we employ a Multilayer Perceptron (MLP) comprising three hidden layers, each containing 50

neurons. ReLU serves as the activation function between each hidden layer. Both the input and output layers have three neurons. The batch size for training in each case is set to 400. The hyperparameter λ for each regularization method (as defined in Eq.2.2) is determined through a grid-search approach.

4.3.1. Hyperparameter analysis

In investigating the influence of changes in subnetwork size on learning capacity in the regression problem, two key hyperparameters come into discussion:

- **Pruning Iterations:** This parameter dictates the number of pruning cycles performed on each subnetwork following its initial pretraining phase.
- α_{fc}: **Pruning Ratio for Fully Connected Layers:** This parameter regulates the fraction of parameters retained within the fully connected layers of each subnetwork. A higher value of α_{fc} implies that a greater number of parameters are preserved, resulting in larger subnetworks.

The number of Pruning Iterations directly influences the size of each subnetwork, with more iterations leading to smaller subnetworks. This outcome arises because a larger number of parameters are pruned during each pruning cycle. Conversely, the value of α_{fc} also impacts subnetwork size. A higher α_{fc} results in larger subnetworks by retaining more parameters in the fully connected layers, while a lower α_{fc} makes the subnetworks smaller. Both of these hyperparameters are of utmost importance as they collectively define the size and complexity of the subnetworks.

4.3.2. Parallel Composite dataset

The Parallel dataset is composed of three tasks from three materials with different continuous hyperelastic constitutive relationships. The hyperelasticity relationship for input stress σ and ϵ is represented in Fig.4.2. As can be seen from the figure, the three composites, especially composite 2, have different hyperelasticity relationships. Therefore, in the case of continuous learning, the neural network model needs to perform continuous learning from task sequences of three changing task types. This is used to test the learning of the model when changing the task sequence.

The grid-search for Pruning Iterations and α_{fc} is represented in Fig.4.3. In this plot, each node corresponds to an experiment conducted during the respective pruning iteration and with a specific value of α_{fc}. The size and number displayed on each node represent the average relative error obtained after the model completes its learning process for all three tasks. Smaller node sizes and numbers indicate better learning performance. Additionally, the color of each node signifies the average task-specific ratio achieved by the model after completing all the task sequences. Deeper node colors indicate smaller task-specific ratios, suggesting that more parameters are shared among different subnetworks.

In the context of the pruning process, the number of pruning iterations plays a critical role in determining the composition of subnetworks. As pruning iterations

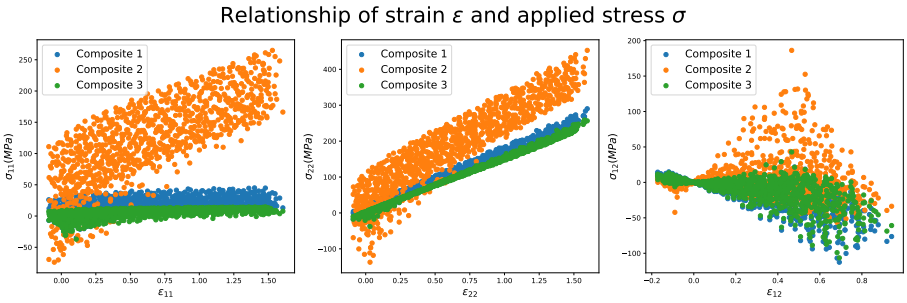


Figure 4.2: Hyperelasticity relationship of input stress σ and ϵ for Parallel composites

progress, there is a notable impact on the subnetworks' size and structure. During each pruning iteration, the pruning function identifies and removes parameters that are considered less important for the model's performance. These less important parameters are often shared parameters that contain information from past tasks. They tend to have a lower importance score (IS) because they are not task-specific. Conversely, task-specific parameters, which are trained specifically for a given task, typically have a higher IS. These parameters contribute significantly to the model's output for the corresponding task.

As the number of pruning iterations increases, the model becomes tighter in retaining parameters. It tends to preserve parameters with higher IS, which are often task-specific parameters. This behavior results in a higher task-specific parameter ratio within each subnetwork. In other words, a larger proportion of the remaining parameters in each subnetwork is dedicated to the specific tasks the model has learned. These retained parameters are well-suited to the tasks, as they have demonstrated their importance through training. In scenarios where α_{fc} is set to a higher value, the subnetworks tend to be larger. This is because a higher α_{fc} value makes the pruning process less stringent, allowing more parameters to be retained. Initially, one might think that this would result in a higher proportion of task-specific parameters within each subnetwork. However, the data shows a different trend: the task-specific ratio decreases.

This outcome can be explained by considering the nature of the parameters that are retained. When α_{fc} is high, the pruning algorithm is less aggressive. This leniency allows not just the task-specific parameters, but also a significant number of shared parameters to survive the pruning process. These shared parameters are usually designed to capture more general features and are less specialized for any single task. Because of this, the larger subnetworks end up having a mix of both task-specific and shared parameters. The presence of these shared parameters dilutes the concentration of task-specific parameters. Consequently, the ratio of task-specific parameters to the total number of parameters in the subnetwork decreases. This leads to a lower task-specific ratio, which is contrary to what one might initially expect from a larger subnetwork.

The task-specific parameter ratio within subnetworks is a crucial aspect of con-

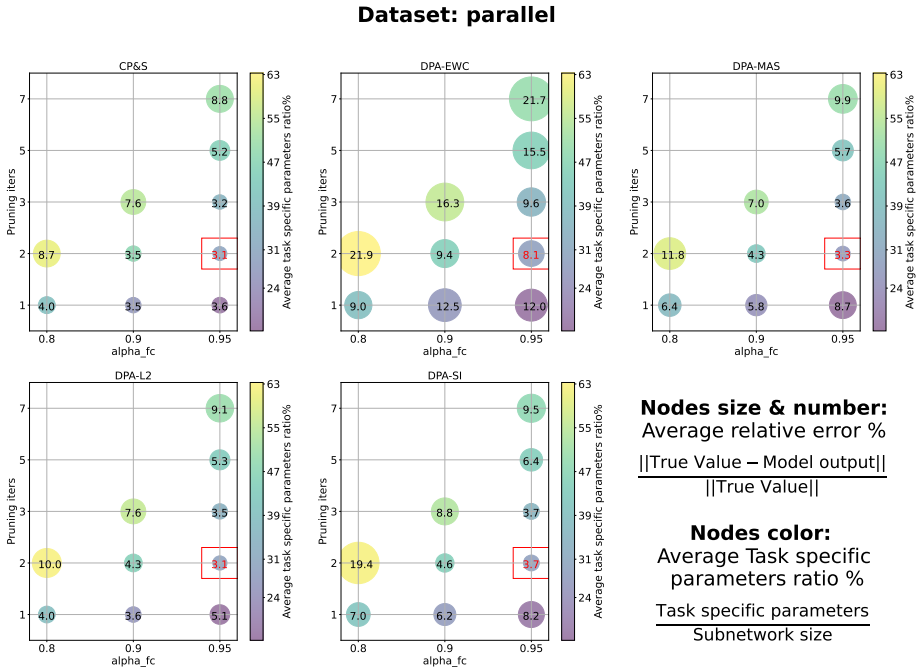
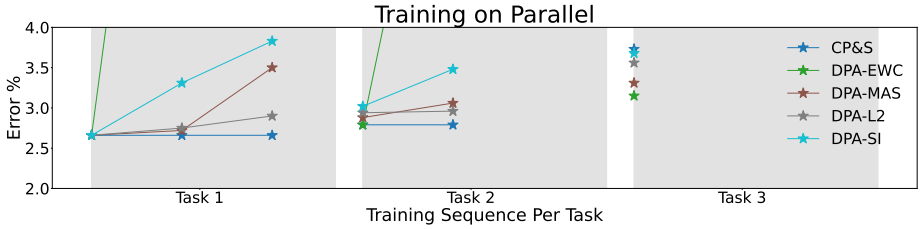


Figure 4.3: Hyperparameter analysis for Parallel dataset. In this plot, the size and the number of the circle represents the average error for each model in the task sequence, smaller size number represents the lower average error for the model. The color represents the tendency of the average task-specific parameters ratios, the deeper the color, the less the average task-specific parameters ratios.

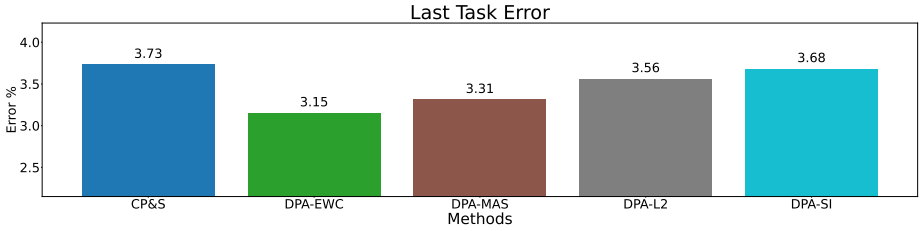
tinual learning models. It is a fine balance that must be maintained to ensure the effective retention of information from both old and new tasks. If the task-specific parameters occupy a significant portion of the subnetworks, it means that there are fewer shared parameters available to capture knowledge from previous tasks. This can result in increased forgetting, where the model loses previously learned information as it focuses more on the new tasks. On the other hand, if the task-specific parameters are too restricted within the subnetworks, the subnetworks may not have sufficient capacity to accommodate the knowledge required for learning new tasks effectively. The challenge lies in finding the right equilibrium where the task-specific parameters are neither too dominant nor too limited. In the context of the pruning iterations and α_{fc} hyperparameter, the aim is to ensure that the subnetworks can effectively balance the retention of old task information with the ability to learn new tasks. It has been observed that the most optimal configuration, leading to the lowest average relative error, occurs when the pruning iteration is set to 2, and α_{fc} is set to 0.95 in the Parallel dataset. This suggests that, in this specific dataset and experimental setup, a moderate task-specific parameter ratio, achieved through these hyperparameters, strikes the best balance between preserving prior knowledge and facilitating new learning.

Fig.4.4 illustrates how the average relative error changes throughout the task

sequence. In contrast to classification problems, regression problems involve predicting continuous values, making it challenging to achieve perfect matches with target results[68]. Consequently, any alterations to the model's task-related information visibly affect its prediction accuracy. The graph reveals a noticeable upward trend in the average relative error for the DPA models. This trend suggests that, as shared parameters are released when transitioning to new tasks, the model's grasp of information from previous tasks undergoes noticeable changes, resulting in evident forgetting of the old task. In Fig.4.5b, all DPA models exhibit lower errors for the final task, indicating that releasing shared parameters enhances the model's performance in learning new tasks.



(a) Relative error for all observation tasks sequence

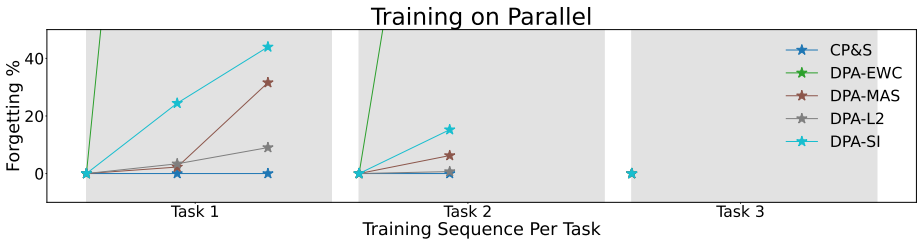


(b) Relative error after learning the final task

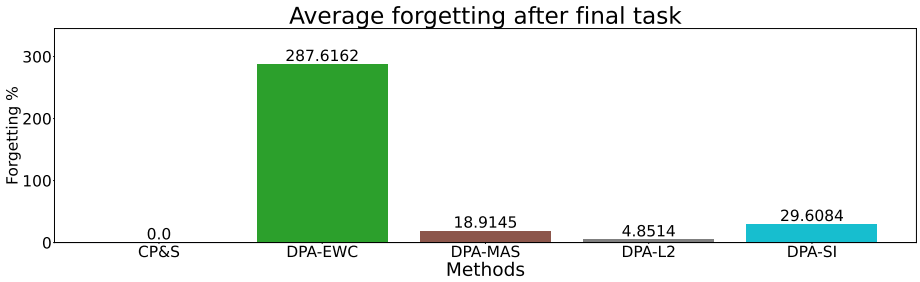
Figure 4.4: The relative error in the Parallel dataset experiment

The extent of forgetting observed in various models is visualized in Fig.4.5. Notably, CP&S maintains a constant forgetting rate throughout, showing no variation in its memory retention. Conversely, the DPA model exhibits an increasing forgetting rate with the accumulation of tasks, signifying that as the model learns more tasks, its ability to retain knowledge from previous tasks diminishes. The consistently positive forgetting rate depicted in the graph implies a lack of significant positive transfer. In Fig.4.5b, DPA-L_2 stands out with the lowest average forgetting rate across the task sequence, averaging only 5.18% forgetting for each task. In contrast, EWC exhibits the highest forgetting, with a substantial loss of information in the old tasks, resulting in a staggering forgetting rate of 277.47%.

For the average relative error, DPA-EWC has the worst error rate, primarily due to its elevated forgetting rate. Conversely, DPA-MAS and DPA-REG exhibit error rates that closely resemble those of CP&S. This suggests that, in the context of regression problems, the DPA framework enhances the model's overall learning capacity throughout the task sequence by expanding the network parameter set



(a) Average forgetting rate for all observation tasks sequence



(b) Average forgetting rate after learning the final task

Figure 4.5: The forgetting rate in the Parallel dataset experiment

available for acquiring new task knowledge, even in the presence of forgetting.

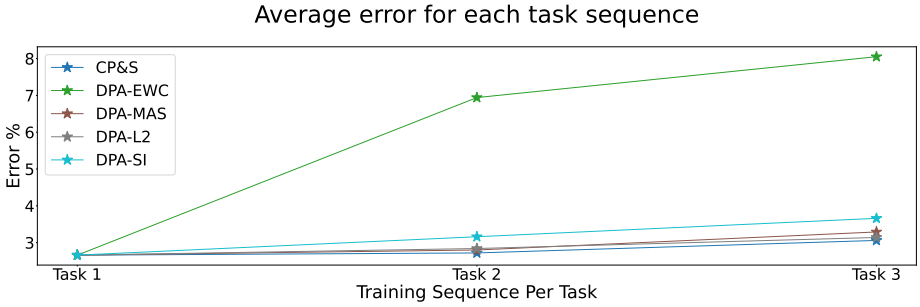


Figure 4.6: The average relative error for all observation tasks in the Parallel dataset experiment

In the Parallel dataset experiment, different tasks exhibit varying ratios of task-specific parameters (see in Appendix.C.1). Initially, the first task has a relatively high ratio, approximately 23-24%. However, as subsequent tasks are introduced, this ratio gradually decreases. For instance, the second task displays a lower ratio of about 8-9%, and the final task stabilizes at a ratio of approximately 16-17%. This phenomenon arises because each subnetwork initially possesses a larger proportion of task-specific parameters. Yet, as additional tasks are integrated, the model adapts by diminishing its reliance on task-specific parameters in later tasks. Fig. 4.4b provides insight into the subnetwork sizes. Each subnetwork constitutes approximately 37% of the total parameters. Upon completing training on the entire

task sequence, 60% of the parameters are utilized across the three subnetworks. Notably, the model's sparsity in the Parallel dataset experiment is comparatively lower than that observed in MNIST-like datasets.

4.3.3. Perpendicular Composite dataset

The relationship between input stress σ and strain ϵ for Perpendicular composites, which follows a hyperelastic behavior, is depicted in Fig.4.7. Analogous to the case with Parallel composites, these three distinct composites demonstrate varied hyperelastic properties. In tasks involving the Perpendicular dataset, the deep learning model is set to undergo training within a continual learning framework, tackling a variety of task types that reflect these differences in hyperelastic performance.

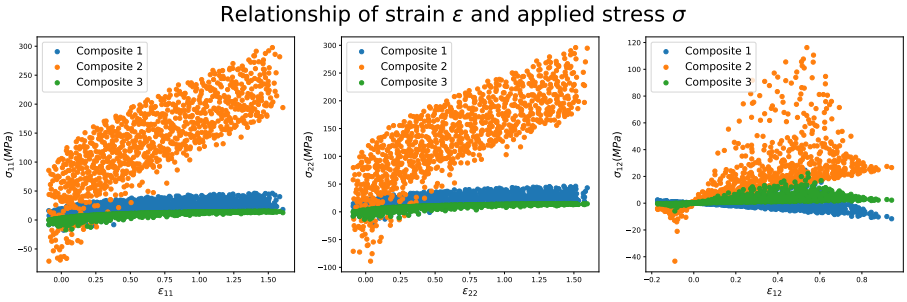


Figure 4.7: Hyperelasticity relationship of input stress σ and ϵ for Perpendicular composites

The analysis of hyperparameters for the Parallel dataset is depicted in Fig. 4.8. Remarkably, the trend observed in the task-specific parameter ratio concerning pruning iterations and α_{fc} is similar to the findings from experiments conducted on the Parallel dataset (as shown in Fig. 4.3). As the number of pruning iterations increases, the task-specific parameter ratio tends to ascend. Conversely, an increment in α_{fc} results in a decreased task-specific parameter ratio. This consistent pattern underscores how these hyperparameters impact the distribution of task-specific parameters across various tasks in both the Parallel and Perpendicular dataset experiments. Notably, for CP&S, DPA-MAS, and DPA-SI, the optimal hyperparameter combination involves 2 pruning iterations with α_{fc} set to 0.95. Conversely, for DPA-EWC and DPA-L_2, the ideal hyperparameter combination consists of 1 pruning iteration with α_{fc} set to 0.9.

The training results for the Perpendicular dataset, as shown in Fig. 4.9, exhibit a trend similar to what was observed in the Parallel dataset experiment. There is a noticeable increase in the average relative error for the DPA models. This increase suggests that when the model transitions to new tasks and releases shared parameters, its retention of information from previous tasks undergoes significant changes, leading to clear evidence of forgetting old tasks. While the error of CP&S has not changed, this is because the parameters for the old information are fixed. For the error on the final task, only DPA-EWC and DPA-MAS show lower errors compared to CP&S. Both DPA-L_2 and DPA-SI perform worse on the last task. Therefore, DPA

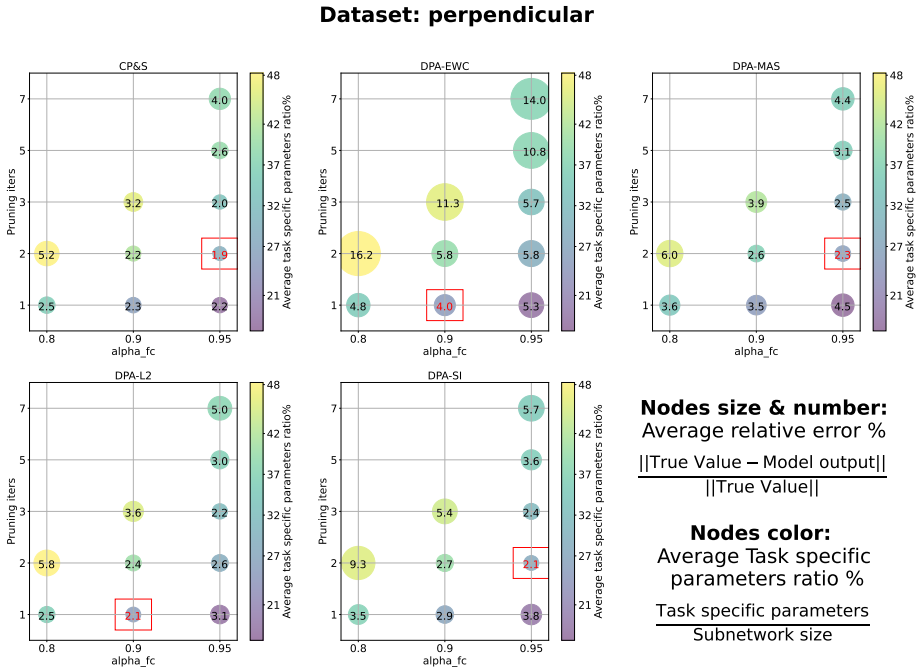


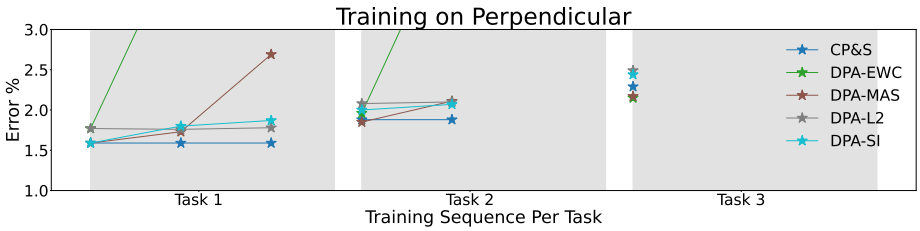
Figure 4.8: Hyperparameter analysis for Perpendicular dataset

still outperforms in learning new tasks when using typical regression methods to update fixed parameters. Note that the errors for the first task vary among different methods. This variation is due to the different hyperparameters (pruning iterations and α_{fc}) chosen for each method.

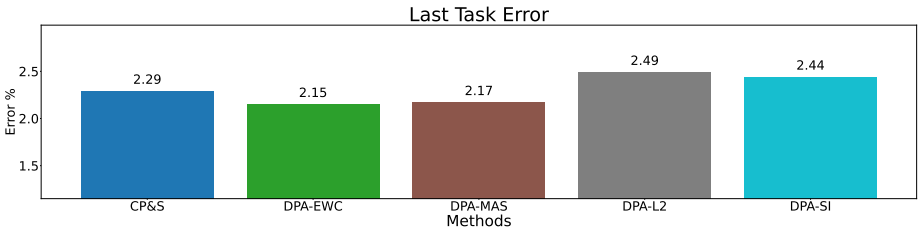
For the analysis of forgetting, as shown in Fig. 4.10, forgetting in DPA methods rises as the task sequence increases. Among all DPA methods, DPA-L_2 shows the lowest rate of forgetting. This is due to the constraint on parameter updates using the L_2 norm. On the other hand, DPA-EWC has the highest rate of forgetting.

Considering the average relative error for all observation tasks in the Perpendicular dataset experiment (in Fig. 4.11), it's evident that CP&S consistently maintained the lowest average relative error for this regression problem. This achievement is attributed to CP&S's ability to retain comprehensive information about past tasks, preventing any instances of forgetting. Although DPA-L_2 also exhibits the smallest forgetting rate, its performance in learning new tasks is lower than CP&S, resulting in a higher average relative error. On the other hand, DPA-MAS, while having a slightly higher average error across the task sequence compared to CP&S, does showcase lower error specifically in the final task. This suggests that even in the presence of some forgetting, DPA-MAS manages to control the overall average error, ensuring it doesn't deviate significantly from CP&S by enhancing its learning capabilities for new tasks.

The sparsity analysis of the model in the Perpendicular Composite dataset is in

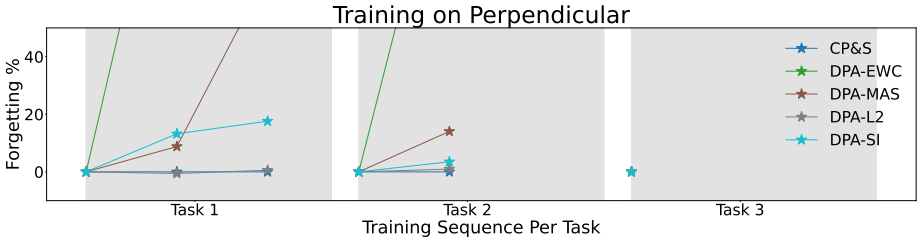


(a) Relative error for all observation tasks sequence

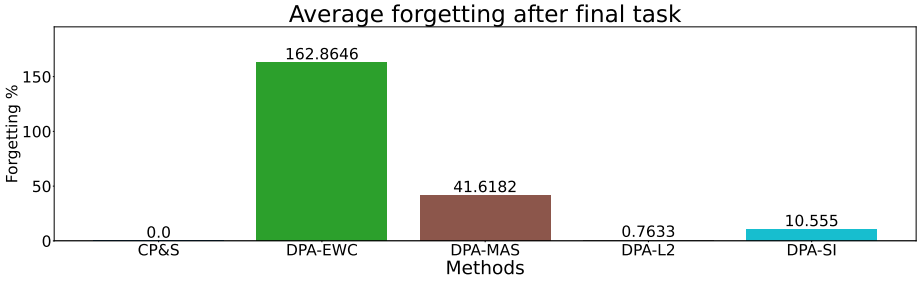


(b) Relative error after learning the final task

Figure 4.9: The relative error in the Perpendicular dataset experiment



(a) Average forgetting rate for all observation tasks sequence



(b) Average forgetting rate after learning the final task

Figure 4.10: The forgetting rate in the Perpendicular data experiment

the Appendix.C.2. The observed differences in task-specific parameter ratios and network sparsity among various models in the Perpendicular dataset experiment can be attributed to variations in pruning iterations and α_{fc} hyperparameters. Specifically, models like DPA-EWC and DPA-L_2 employ lower pruning iterations and α_{fc}

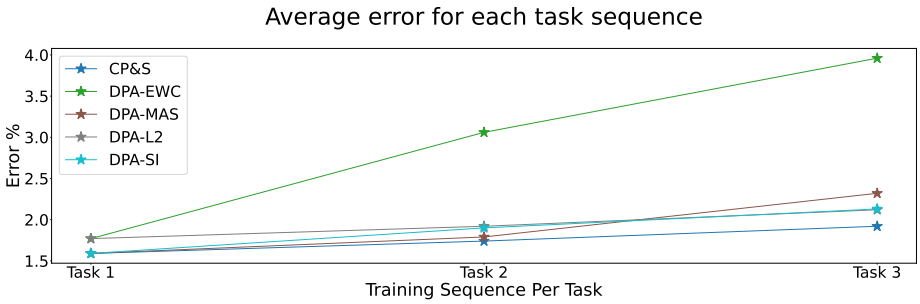


Figure 4.11: The average relative error for all observation tasks in the Perpendicular dataset experiment

values, which means they retain more task-specific parameters during the pruning process. Consequently, this leads to larger subnetworks and a higher union subnetwork size when compared to other models. However, the distribution of task-specific parameters within these larger subnetworks follows a unique pattern. In the first subnetwork, the task-specific parameter ratio for DPA-EWC and DPA-L_2 is lower than that of other models. This suggests that these models initially allocate fewer parameters to task-specific information in the first subnetwork. Yet, the situation changes in the second and third subnetworks. For DPA-EWC and DPA-L_2, the task-specific parameter ratio becomes higher than that of other models in these subnetworks, with the exception of the third subnetwork for CP&S. This phenomenon can be explained by the models' attempt to adapt to different task requirements. Initially, they prioritize shared parameters over task-specific ones in the first subnetwork. However, as they progress to the second and third subnetworks, the models recognize the need to allocate more parameters to task-specific information for better task performance. This adaptability in parameter allocation within subnetworks contributes to the observed variations in task-specific parameter ratios.

4.3.4. Mix dataset

To further test the continuous learning capabilities across task types of the model, the Mix dataset is constructed by combining both Parallel and Perpendicular datasets. The Mix dataset comprises three tasks from the Parallel dataset followed by three tasks from the Perpendicular dataset. This arrangement introduces a significant shift in the task types and complexities encountered during training. As the neural networks transition from the initial three tasks to the subsequent three, they are required to adapt to and learn from tasks of varying types and characteristics.

To evaluate how effectively the Dynamic Parameters Architectural (DPA) method adapts to this shift in task type, we examine the relative error across the task sequence. Fig. 4.12 demonstrates how different pruning hyperparameters impact the average relative error and the average task-specific parameter ratio throughout the task sequence. Notably, the observed trend in the task-specific parameter ratio, concerning both pruning iterations and α_{fc}, aligns with the patterns identified in the

experiments conducted on the Parallel and Perpendicular datasets (as depicted in Fig. 4.3 and Fig. 4.8). Specifically, an increase in the number of pruning iterations tends to raise the task-specific parameter ratio, while an elevation in α_{fc} leads to a decrease in the task-specific parameter ratio. For CP&S, DPA-MAS and DPA-SI, the lowest relative error occurs when pruning iterations and α_{fc} are 3 and 0.95, respectively. For DPA-EWC, the pruning iterations and α_{fc} are 2 and 0.9. For DPA-L_2 the best performance occurs lowest pruning iterations and α_{fc}.

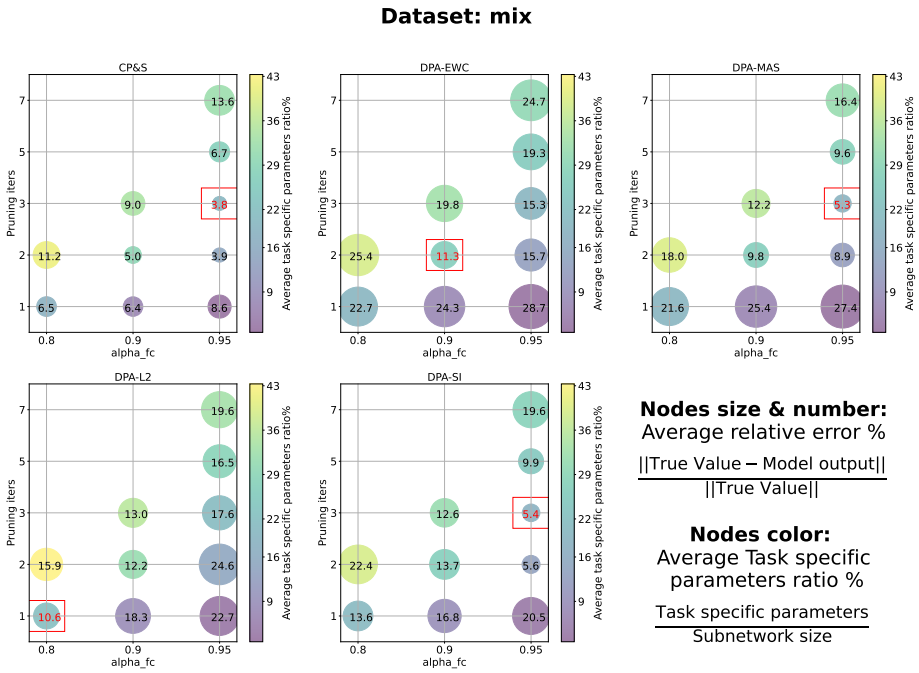
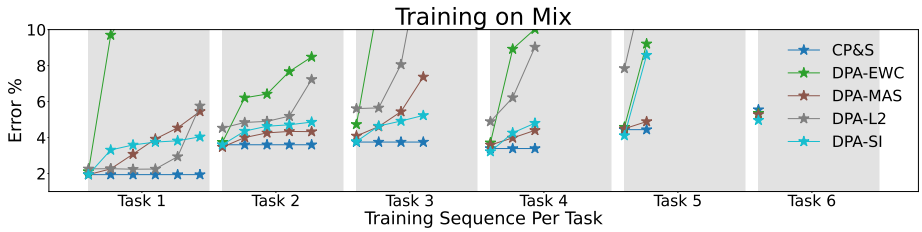


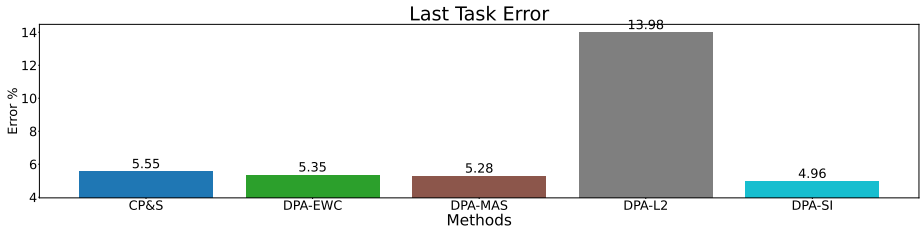
Figure 4.12: Hyperparameter analysis for Mix dataset

Fig. 4.13 displays the relative error for each model across the task sequence. As the number of tasks increases and the task complexity shifts, all DPA models exhibit noticeable forgetting of previous tasks. Notably, CP&S maintains consistent performance in the old tasks, while other methods experience an obvious increase in relative error for these tasks. In contrast, for the final task (as shown in Fig. 4.13b), DPA-EWC, DPA-MAS, and DPA-SI exhibit lower performance compared to CP&S, implying that the DPA models excel in learning new tasks. However, DPA-L_2 stands out in this experiment, displaying significant forgetting of previous tasks and a high error rate in the final task. This is because the L_2 regularization method focuses on the reusability of parameters [64], which limits the update of each shared parameter on the new task, rather than just the important parameters, which makes the new network learn slower on the learning of the new task, especially if the task type changes, so the learning ability of the model in the new task decreases

The analysis of forgetting for the task sequence is presented in Fig. 4.14. As the



(a) Relative error for all observation tasks sequence



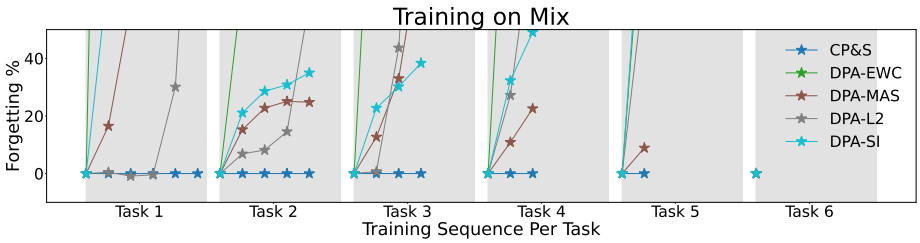
(b) Relative error after learning the final task

Figure 4.13: The relative error in the Mix dataset experiment

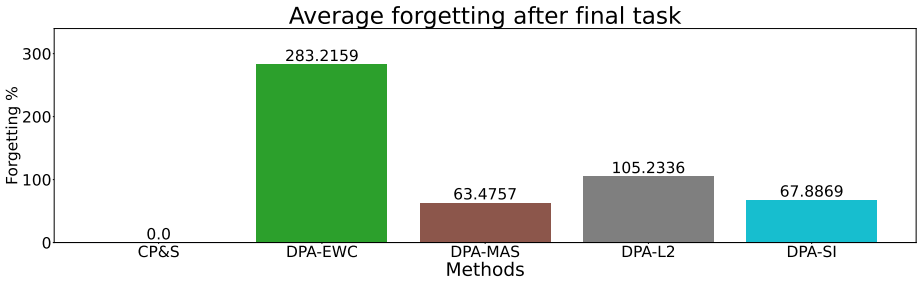
number of tasks increases and their complexity rises, each DPA model experiences an elevated forgetting rate compared to previous datasets. Among all DPA models, DPA-EWC consistently exhibits the highest rate of forgetting. In contrast, DPA-MAS and DPA-SI maintain the lowest rates of forgetting. Notably, DPA-L_2 shows a high average forgetting rate across the entire task sequence. For further clarification, Fig. 4.14a reveals that the forgetting rate for the first three tasks is less than 20%. This rate is lower than those observed for DPA-MAS and DPA-SI. However, upon the introduction of the fourth task from the Perpendicular dataset, the forgetting rates for tasks 1 and 2 increase abruptly. This sudden change contributes to an overall elevation in the average forgetting rate for all tasks.

The average relative error for various models in the Mix data experiment is depicted in Fig.4.14. Due to the heterogeneity of task types, parameter reuse adversely impacts the model's performance. It leads to severe forgetting, causing DPA-L_2 to perform poorly on this dataset. The average errors for DPA-MAS and DPA-SI are quite similar. Their average relative errors in the final task sequence are approximately 2% higher than those of CP&S. Given that DPA-SI demonstrates a more better learning ability than CP&S in the final task, it can be concluded that DPA models can provide a balance between learning capacity in new tasks and forgetting. This equilibrium is achieved by enhancing the model's learning capacity for new tasks while allowing for some degree of forgetting.

The sparsity levels for various models are depicted in Fig.C.3 (see in Appendix.C.3. DPA-EWC stands out for having high task-specific parameter ratios but relatively small subnetwork sizes. This phenomenon can be attributed to its low setting for α_{fc}. With a lower α_{fc} and higher pruning iteration, DPA-EWC tends to retain more task-specific parameters, resulting in higher task-specific parameter ratios. By the



(a) Average forgetting rate for all observation tasks sequence



(b) Average forgetting rate after learning the final task

Figure 4.14: The forgetting rate in the Mix data experiment

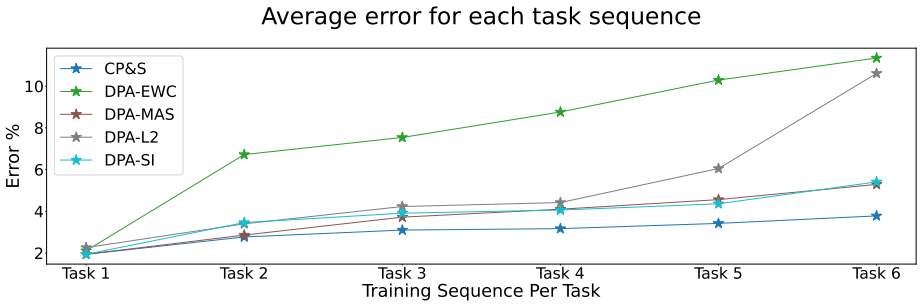


Figure 4.15: The average relative error for all observation tasks in the Mix dataset experiment

end of the task sequence, these ratios for the first subnetwork range between 40% and 60%. However, it's important to note that all DPA models exhibit some degree of forgetting for task 1. This suggests that the parameters most influential for task 1 have likely undergone significant modifications. As a result, the model's information for task 1 has been altered. In contrast, DPA-L_2 shows the lowest sparsity among the models, despite its low α_{fc} setting. This is primarily due to fewer pruning iterations, which leads to the retention of more parameters. The remaining models generally exhibit sparsity levels around 40%, indicating a balanced trade-off between task-specific and shared parameters.

4.4. Discussions

- **Learning Capabilities:** The regularization methods used in DPA provide a trade-off between learning new tasks and forgetting old ones. Under the premise of allowing some level of forgetting, DPA enhances its learning capacity for new RVEs.
- **Balance between forgetting and saturation:** Owing to the high forgetting for the previous information DPA model can not surpass the overall learning performance of CP&S, but is close to CP&S. However, due to the improvement of the learning capacity in the new task. The new methods still balance the learning capacity of new tasks and the anti-forgetting capacity.
- **Regularization for Complex RVEs types:** For a wide range of non-correlated RVE modeling tasks, parameter-focused regularization methods like SI and MAS appear more fitting with DPA. They aim to preserve important parameters for individual RVE modeling. This approach helps the model adapt better to new task challenges, keeping previously learned patterns intact.
- **Forgetting Resistance:** The performance for the DPA model doesn't have a direct relationship with the task-specific parameters. However, When the model has fewer task-specific parameters, it tends to have a high resistance to catastrophic forgetting, resulting in a lower average modeling error rate.

5

Conclusion

In this study, we have introduced the Dynamic Parameters Architectural (DPA) method as an evolution from the CP&S approach. The primary innovation lies in how DPA manages parameters within overlapping subnetworks. Unlike CP&S, which is limited to task-specific parameters, DPA updates shared parameters between subnetworks. By freeing up shared parameters among subnetworks, DPA has been designed to provide a more adaptable and flexible mechanism compared to CP&S.

Our findings directly address the concerns raised in Knowledge Gap 1. Specifically, the DPA method provides a solution to the issue of network saturation commonly found in architectural methods. By enabling the dynamic adaptation of parameters within overlapping subnetworks, DPA mitigates the saturation challenge, demonstrating improved learning performance in our experiments when compared to CP&S. In addressing **Knowledge Gap 2**, we have empirically demonstrated that regularization-based continual learning algorithms update only parts of a network. This helps in resolving the network saturation issue. The incorporation of L_2 norm regularization within DPA, especially in classification continual learning scenarios, effectively preserves information from past tasks while simultaneously adapting to new ones. Moreover, our research also addresses **Knowledge Gap 3**. By leveraging DPA, we ensure that information from prior tasks is adeptly stored within the structure of subnetworks. This not only enhances the reusability of parameters but also crucially avoids the introduction of any additional parameters, negating the need for extra storage space—an advancement than the CP&S.

Experiments for classification continual learning were conducted using the widely recognized MNIST, Fashion-MNIST, and EMNIST datasets. For regression continual learning scenarios, datasets focusing on the hyperelastic properties of composites, specifically those with perpendicular and parallel fibers, were employed. This selection ensured a comprehensive evaluation of the DPA method across both classification and regression tasks.

Key findings and observations from our experiments include:

- DPA demonstrates superior learning capabilities in continual learning scenarios by providing balance to the acquisition of new tasks and the retention of old tasks. Compared with CP&S, DPA has a higher learning performance in classification continual learning.
- The use of L_2 norm regularization in DPA has proven particularly effective for classification continual learning, ensuring information preservation and adaptability.
- For data-driven modeling scenarios, methods like DPA-SI and DPA-MAS, which focus on the importance of specific parameter distribution, are more suitable for complex RVE types. These methods illustrate a close learning capacity with the CP&S, but higher performance in the new task.
- Models with fewer task-specific parameters exhibit better retention capabilities. This emphasizes the value of shared parameters applicable across diverse tasks or RVEs.

In conclusion, the Dynamic Parameters Architectural (DPA) method effectively resolves the issue of network saturation while striking a balance between preserving prior knowledge and enhancing learning capacity. It outperforms CP&S in achieving these objectives. Future research can further explore its potential and applicability across a broader spectrum of application scenarios, ensuring the ongoing efficiency and resilience of continual learning techniques in the context of ever more intricate data and tasks.

Bibliography

[1] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[2] Robert M French. "Catastrophic forgetting in connectionist networks". In: *Trends in cognitive sciences* 3.4 (1999), pp. 128–135.

[3] Michael McCloskey and Neal J Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.

[4] Matthias De Lange et al. "A continual learning survey: Defying forgetting in classification tasks". In: *IEEE transactions on pattern analysis and machine intelligence* 44.7 (2021), pp. 3366–3385.

[5] Mitchell Wortsman et al. "Supermasks in superposition". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 15173–15184.

[6] Ghada Sokar, Decebal Constantin Mocanu, and Mykola Pechenizkiy. "Spacenet: Make free space for continual learning". In: *Neurocomputing* 439 (2021), pp. 1–11.

[7] Aleksandr Dekhovich et al. "Neural network relief: a pruning algorithm based on neural activity". In: *arXiv preprint arXiv:2109.10795* (2021).

[8] Ian J Goodfellow et al. "An empirical investigation of catastrophic forgetting in gradient-based neural networks". In: *arXiv preprint arXiv:1312.6211* (2013).

[9] Steven A Siegelbaum and Eric R Kandel. "Learning-related synaptic plasticity: LTP and LTD". In: *Current opinion in neurobiology* 1.1 (1991), pp. 113–120.

[10] Judith E Dayhoff and James M DeLeo. "Artificial neural networks: opening the black box". In: *Cancer: Interdisciplinary International Journal of the American Cancer Society* 91.S8 (2001), pp. 1615–1635.

[11] Rupesh K Srivastava et al. "Compete to compute". In: *Advances in neural information processing systems* 26 (2013).

[12] Piyabute Fuangkhon and Thitipong Tanprasert. "An incremental learning algorithm for supervised neural network with contour preserving classification". In: *2009 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*. Vol. 2. IEEE. 2009, pp. 740–743.

[13] Fan Feng et al. "Challenges in task incremental learning for assistive robotics". In: *IEEE Access* 8 (2019), pp. 3434–3441.

[14] Guy Oren and Lior Wolf. "In defense of the learning without forgetting for task incremental learning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 2209–2218.

[15] Gido M Van de Ven and Andreas S Tolias. "Three scenarios for continual learning". In: *arXiv preprint arXiv:1904.07734* (2019).

[16] Zhizhong Li and Derek Hoiem. "Learning without forgetting". In: *IEEE transactions on pattern analysis and machine intelligence* 40.12 (2017), pp. 2935–2947.

[17] Sylvestre-Alvise Rebuffi et al. "icarl: Incremental classifier and representation learning". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 2001–2010.

[18] Gyuhak Kim et al. "A Theoretical Study on Solving Continual Learning". In: *arXiv preprint arXiv:2211.02633* (2022).

[19] Jathushan Rajasegaran et al. "itaml: An incremental task-agnostic meta-learning approach". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 13588–13597.

[20] Aleksandr Dekhovich et al. "Continual prune-and-select: class-incremental learning with specialized subnetworks". In: *Applied Intelligence* (2023), pp. 1–16.

[21] Anthony Robins. "Catastrophic forgetting, rehearsal and pseudorehearsal". In: *Connection Science* 7.2 (1995), pp. 123–146.

[22] Andrei A Rusu et al. "Progressive neural networks". In: *arXiv preprint arXiv:1606.04671* (2016).

[23] Arun Mallya and Svetlana Lazebnik. "Packnet: Adding multiple tasks to a single network by iterative pruning". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2018, pp. 7765–7773.

[24] Abhishek Aich. "Elastic weight consolidation (EWC): Nuts and bolts". In: *arXiv preprint arXiv:2105.04093* (2021).

[25] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. "Variational inference: A review for statisticians". In: *Journal of the American statistical Association* 112.518 (2017), pp. 859–877.

[26] Rahaf Aljundi et al. "Memory aware synapses: Learning what (not) to forget". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 139–154.

[27] Friedemann Zenke, Ben Poole, and Surya Ganguli. "Continual learning through synaptic intelligence". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3987–3995.

[28] David Rolnick et al. "Experience replay for continual learning". In: *Advances in Neural Information Processing Systems* 32 (2019).

[29] Arslan Chaudhry et al. "Continual learning with tiny episodic memories". In: (2019).

[30] Yue Wu et al. "Large scale incremental learning". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 374–382.

[31] Arthur Douillard et al. "Podnet: Pooled outputs distillation for small-tasks incremental learning". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*. Springer. 2020, pp. 86–102.

[32] Minsoo Kang, Jaeyoo Park, and Bohyung Han. "Class-incremental learning by knowledge distillation with adaptive feature consolidation". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 16071–16080.

[33] Pietro Buzzega et al. "Dark experience for general continual learning: a strong, simple baseline". In: *Advances in neural information processing systems* 33 (2020), pp. 15920–15930.

[34] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[35] Sang-Woo Lee et al. "Overcoming catastrophic forgetting by incremental moment matching". In: *Advances in neural information processing systems* 30 (2017).

[36] Junting Zhang et al. "Class-incremental learning via deep model consolidation". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 1131–1140.

[37] Larry R Squire. "Memory and brain". In: (1987).

[38] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. "Expert gate: Lifelong learning with a network of experts". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 3366–3375.

[39] Siavash Golkar, Michael Kagan, and Kyunghyun Cho. "Continual learning via neural pruning". In: *arXiv preprint arXiv:1903.04476* (2019).

[40] Song Han et al. "Learning both weights and connections for efficient neural network". In: *Advances in neural information processing systems* 28 (2015).

[41] Zhuang Liu et al. "Rethinking the value of network pruning". In: *arXiv preprint arXiv:1810.05270* (2018).

[42] Yoshio Izui and Alex Pentland. "Analysis of neural networks with redundancy". In: *Neural Computation* 2.2 (1990), pp. 226–238.

[43] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[44] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[45] Zifeng Wang et al. "Learn-prune-share for lifelong learning". In: *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2020, pp. 641–650.

[46] Jaehong Yoon et al. "Lifelong learning with dynamically expandable networks". In: *arXiv preprint arXiv:1708.01547* (2017).

[47] Yang Yang, Bo Chen, and Hongwei Liu. "Bayesian compression for dynamically expandable networks". In: *Pattern Recognition* 122 (2022), p. 108260.

[48] Fu-Yun Wang et al. "Foster: Feature boosting and compression for class-incremental learning". In: *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXV*. Springer. 2022, pp. 398–414.

[49] David Lopez-Paz and Marc'Aurelio Ranzato. "Gradient episodic memory for continual learning". In: *Advances in neural information processing systems* 30 (2017).

[50] Hao Wang and Dit-Yan Yeung. "A survey on Bayesian deep learning". In: *ACM Computing Surveys (CSUR)* 53.5 (2020), pp. 1–37.

[51] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[52] Max Kochurov et al. "Bayesian incremental learning for deep neural networks". In: *arXiv preprint arXiv:1802.07329* (2018).

[53] Ananth Ranganathan. "Assumed density filtering". In: *Georgia Inst. of Technology, Atlanta, GA* (2004).

[54] Zhenming Shun and Peter McCullagh. "Laplace approximation of high dimensional integrals". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 57.4 (1995), pp. 749–760.

[55] Agustinus Kristiadi. *Fisher Information Matrix*. https://agustinus.kristia.de/techblog/2018/03/11/fisher-information/.

[56] Cuong V Nguyen et al. "Variational continual learning". In: *arXiv preprint arXiv:1710.10628* (2017).

[57] Malay Ghosh and Ruitao Liu. "Moment matching priors". In: *Sankhya A* 73.2 (2011), pp. 185–201.

[58] Yen-Chang Hsu et al. "Re-evaluating continual learning scenarios: A categorization and case for strong baselines". In: *arXiv preprint arXiv:1810.12488* (2018).

[59] Yann LeCun. "The MNIST database of handwritten digits". In: *http://yann.lecun. com/exdb/mnist/* (1998).

[60] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms". In: *arXiv preprint arXiv:1708.07747* (2017).

[61] Gregory Cohen et al. "EMNIST: Extending MNIST to handwritten letters". In: *2017 international joint conference on neural networks (IJCNN)*. IEEE. 2017, pp. 2921–2926.

[62] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[63] Jonathan Schwarz et al. "Progress & compress: A scalable framework for continual learning". In: *International conference on machine learning*. PMLR. 2018, pp. 4528–4537.

[64] Xilai Li et al. "Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3925–3934.

[65] Pin Zhang, Zhen-Yu Yin, and Yin-Fu Jin. "State-of-the-art review of machine learning applications in constitutive modeling of soils". In: *Archives of Computational Methods in Engineering* (2021), pp. 1–26.

[66] Miguel A Bessa et al. "A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality". In: *Computer Methods in Applied Mechanics and Engineering* 320 (2017), pp. 633–667.

[67] G Abaqus. "Abaqus 6.11". In: *Dassault Systemes Simulia Corporation, Providence, RI, USA* (2011), p. 3.

[68] Michael LeBlanc and Robert Tibshirani. "Combining estimates in regression and classification". In: *Journal of the American Statistical Association* 91.436 (1996), pp. 1641–1650.

[69] Charles Blundell et al. "Weight uncertainty in neural network". In: *International conference on machine learning*. PMLR. 2015, pp. 1613–1622.

[70] Thang Bui et al. "Deep Gaussian processes for regression using approximate expectation propagation". In: *International conference on machine learning*. PMLR. 2016, pp. 1472–1481.

[71] Jacob Goldberger and Sam Roweis. "Hierarchical clustering of a mixture model". In: *Advances in neural information processing systems* 17 (2004).

[72] Kai Zhang and James T Kwok. "Simplifying mixture models through function approximation". In: *IEEE Transactions on Neural Networks* 21.4 (2010), pp. 644–658.

[73] Surajit Ray and Bruce G Lindsay. "The topography of multivariate normal mixtures". In: *The Annals of Statistics* 33.5 (2005), pp. 2042–2065.

[74] David JC MacKay. "A practical Bayesian framework for backpropagation networks". In: *Neural Computation* 4.3 (1992). One Rogers Street, Cambridge, MA 02142-1209, USA, pp. 448–472.

A

Appendix to Chapter 2

A.1. Progressive Neural Networks (progressive network)

The accomplishment of the progressive network[22] is very concise. After the model learned a task, the parameters of the model on this task are fixed and stored as the previous network. Whenever a new task reaches the model, a new column (a new network) is generated to learn the new data. To make use of the previous information in the updating of the parameters on new tasks, data from the new task also be input into all previous networks, and input the output of each layer of the previous network together with the output of each layer of the current network into the next layer to train the current network. Specifically, a model that generates task T is expressed as shown in Eq.A.1. The hidden layer $h_t^{(T)}$ of the column T(on task T) and layer i gets output from the weight $W_t^{(T)}$ times hidden layer $h_{i-1}^{(T)}$ on task T and the hidden layers from all previous networks $\sum_j^T U_t^{(j:T)} h_{i-1}^{(j)}$. The transverse connection weight between column j to column T ($U_t^{(j:T)}$) controls the signal strength of the old information. In this expression, f is the activation function, which is $f(x) = max(0, x)$. The workflow of the progressive network is as follows (Fig.A.1):

$$h_t^{(T)} = f\left(W_t^{(T)} h_{i-1}^{(T)} + \sum_j^T U_t^{(j:T)} h_{i-1}^{(j)}\right) \tag{A.1}$$

In general, the progressive network is to extract the information of the previous neural network and merges it with the current input information, during the new network training. The advantage of the progressive network is that it fixes the parameters from old tasks, preventing the catastrophic forgetting brought by the change of parameters update. Besides, knowledge from the previous tasks also transfers to the learning in new tasks. In the progressive network, knowledge

A

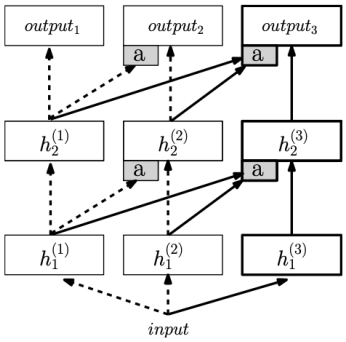


Figure A.1: The parameter update in Progressive network[22].

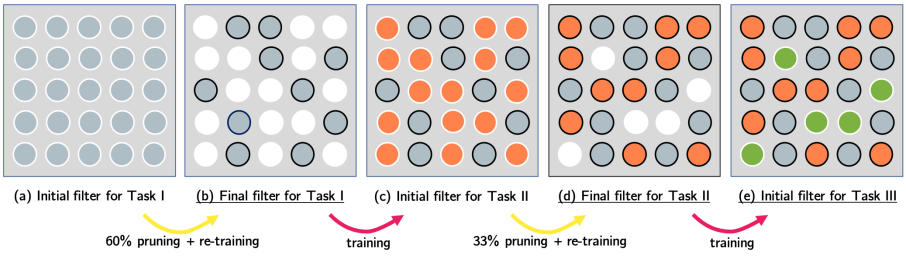


Figure A.2: Solving catastrophic forgetting through Network pruning in PackNet[23]

from old data is not instantaneous, it can be integrated at each level of the feature hierarchy. Furthermore, adding new capacity alongside the progressive network gives these models the flexibility to reuse old computations and learn new tasks. However, the knowledge transfer in the progressive network will not always have a positive effect on the learning process especially if the old tasks are not relevant to the new tasks yet. Meanwhile, the requirement of saving the total model also increases the pressure on the device's memory.

A.2. PackNet

Each training session of a certain task in **PackNet** can be divided into two parts: pre-training and re-training[23]. Pre-training is aim to find which parameters with the highest magnitude are suitable to be saved as the task-specified parameters, other unimportant parameters are pruned by constituting binary masks. Then, the remaining parameters are retained on the given task until convergence. To limit the size of parameters used for every task, a hyperparameter $\mathcal{H}$ is used to control the pre-training mask fraction. The pruning and training session is described in Fig.A.2

Fig.A.3 illustrates the performance of PackNet compared with Learning without Forgetting (LwF)[16] in training in four tasks. The accuracy in PackNet doesn't reduce with the increase in task number. During the training, each certain task-

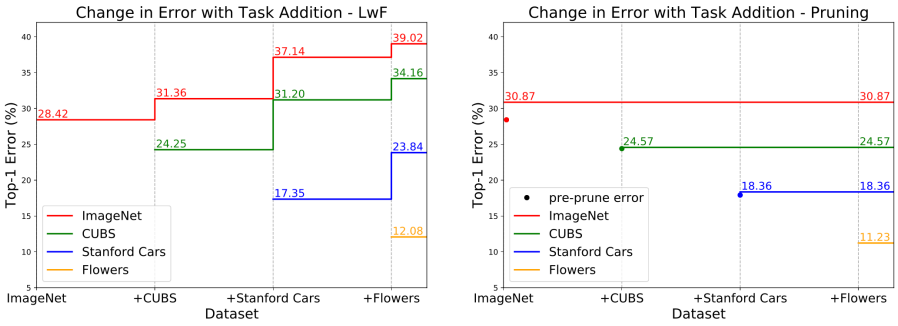


Figure A.3: Performance of PackNet in comparing with Learning without Forgetting[16] in 4 tasks continual learning[23].

specified parameter is fixed, while catastrophic occurs when the parameters shift between old and new tasks, so PackNet will remember every task precisely because of the consolidation of parameters[23]. Is that mean catastrophic forgetting has been solved by PackNet? There is a vital disadvantage of PackNet that the free parameters in a neural network are limited. Once all parameters are assigned to tasks, the model cannot create more parameters for new data[4]. Therefore, the maximum task number learned by PackNet is smaller than other methods (e.g. in progressive network[22], the model can generate infinite models for tasks), unless the size of the model in PackNet is large enough. Nevertheless, increasing the size of the neural network will challenge the device's memory again, which is the same problem faced by the progressive network. Consequently, the way parameters are assigned to each task needs to be redesigned to further reduce the size of the saved model.

A.3. Laplace approximation

Set $f(\theta) = \log P(\theta_{1:T-1}|D_{1:T-1})$, the first step of Laplace approximation is to calculate the second order Taylor extension of the $f(\theta)$ at $\theta = \theta_{1:T-1}$:

$$f(\theta) = f(\theta_{1:T-1}) + \frac{\partial f(\theta)}{\partial \theta}\Big|_{\theta_{1:T-1}} \tag{A.2}$$

$$f(\theta) = f(\theta_{1:T-1}) + \frac{\partial f(\theta)}{\partial \theta}\Big|_{\theta_{1:T-1}} + \frac{1}{2} \cdot \frac{\partial^2 f(\theta)}{\partial \theta^2}\Big|_{\theta_{1:T-1}} (\theta - \theta_{1:T-1})^2 + R(\theta_{1:T-1}) \tag{A.3}$$

Since the model converges on the previous task, $\theta_{1:T-1}$ is the MAP of the parameter of all old tasks, so $P(\theta|D_{1:T-1})$ reaches its maximum point when $\theta = \theta_{1:T-1}$. So, $\frac{\partial f(\theta)}{\partial \theta}\Big|_{\theta_{1:T-1}} = 0$ as well as higher-order terms of the Taylor expansion $R(\theta_{1:T-1})$ can be neglected. The Taylor expansion becomes:

A

$$f(\theta) = f(\theta_{1:T-1}) + \frac{1}{2} \cdot \underbrace{\frac{\partial^2 f(\theta)}{\partial \theta^2}\Big|_{\theta_{1:T-1}}}_{Hessian} (\theta - \theta_{1:T-1})^2 \tag{A.4}$$

Transfer $f(\theta)$ back to $\log P(\theta|D_{1:T-1})$:

$$\log P(\theta|D_{1:T-1}) = \log P(\theta_{1:T-1}|D_{1:T-1}) + \frac{1}{2} \cdot \underbrace{\frac{\partial^2 \log P(\theta|D_{1:T-1})}{\partial \theta^2}\Big|_{\theta_{1:T-1}}}_{Hessian} (\theta - \theta_{1:T-1})^2 \tag{A.5}$$

Eq.A.5 is similar to the expression of Gaussian distribution if the secondary derivative terms (Hessian martix) $\frac{\partial^2 \log P(\theta|D_{1:T-1})}{\partial \theta^2}\Big|_{\theta_{1:T-1}}$ can be written as $\Big(-\frac{\partial^2 \log P(\theta|D_{1:T-1})}{\partial \theta^2}\Big|_{\theta_{1:T-1}}\Big)^{-1}$ and take out log

$$P(\theta|D_{1:T-1}) = P(\theta_{1:T-1}|D_{1:T-1}) \cdot exp\left[\frac{1}{2} \cdot \Big(-\frac{\partial^2 \log P(\theta|D_{1:T-1})}{\partial \theta^2}\Big|_{\theta_{1:T-1}}\Big)^{-1} (\theta - \theta_{1:T-1})^2\right] \tag{A.6}$$

The Eq.A.6 is the Laplace approximation of the intractable distribution $P(\theta|D_{1:T-1})$. With the approximation, since $P(\theta_{1:T-1}|D_{1:T-1})$ is a constant, the posterior pdf $P(\theta|D_{1:T-1})$ is a Gaussian distribution:

$$P(\theta|D_{1:T-1}) \sim \mathcal{N}\left(\theta_{1:T-1}, \Big(-\frac{\partial^2 \log P(\theta|D_{1:T-1})}{\partial \theta^2}\Big|_{\theta_{1:T-1}}\Big)^{-1}\right) \tag{A.7}$$

A.4. Variational continual learning (VCL)

The main idea of variational inference is to use $q(\theta|\phi)$ to fit the intractable pdf $P(\theta|D)$ [25]. Then we set the exponential family distribution with the parameter ϕ:

$$q(\theta|\phi) \sim \mathcal{N}(\phi) \tag{A.8}$$

To approximate the intractable pdf $P(\theta|D)$ (in Eq.2.5), we need to get the distributions q and P as "close" to each other as possible. To solve this problem, we come back to Baye's rule in task T:

$$P(\theta|D_{1:T}) \propto P(\theta)P(D_{1:T-1}|\theta) \tag{A.9}$$

With the approximation, the relationship between $P(\theta|D_{1:T})$ and $q(\theta|(\phi_T)$ is the projection of the right term on the parameters space.

$$\begin{aligned} P(\theta|D_{1:T}) \approx q(\theta|\phi_T) &= proj\Big(P(\theta|D_{1:T-1})P(D_T|\theta)\Big) \\ &= proj\Big(q(\theta|\phi_{T-1})P(D_T|\theta)\Big) \end{aligned} \tag{A.10}$$

To guarantee the manually introduced exponential family distribution $q(\theta|\phi_T)$ can precisely approximate the So, the optimization target is to minimize the KL divergence between these distributions:

$$q(\theta|\phi_T) = \arg\min KL\left[q(\theta|\phi)\middle\|\frac{1}{Z_T}q(\theta|\phi_{T-1})P(D_T|\theta)\right] \tag{A.11}$$

In this equation, Z_t is the intractable normalizing constant that will not influence the final result in MAP of θ. For the first approximate distribution $q(\theta|\phi_1)$ is defined to be the prior of the neural network in the initial period $P(\theta)$. If we extend the KL-divergence term, the express is as follows:

$$\begin{aligned} &KL\left[q(\theta|\phi_T)\middle\|\frac{1}{Z_T}q(\theta|\phi_{T-1})P(D_T|\theta)\right] \\ =&q(\theta|\phi_T)\cdot\ln\frac{q(\theta|\phi_T)}{q(\theta|\phi_{T-1})} - q(\theta|\phi_T)\cdot\ln P(D_T|\theta) + \ln Z_T \end{aligned} \tag{A.12}$$

Now we get the expression for the distribution $q(\theta|\phi_T)$ to approximate $P(\theta|D_{1:T})$. The total loss function of the VCL is obtained:

$$\underbrace{\mathcal{L}_{VCL}(D_T)}_{Total\,loss} = \underbrace{\mathcal{L}(D_T)}_{loss\,on\,Task\,T} \underbrace{- KL\left[q(\theta|\phi_T)\middle\|q(\theta|\phi_{T-1})\right]}_{regularization\,term} \tag{A.13}$$

How to calculate the KL divergence in the regularization term in Eq.A.13 is described in the paper. Researchers constructed a small episodic memory by combining VI with the coreset data[69, 70]. The coreset is like an episodic memory in the replay method to that retains key information from previous tasks.

Input: Prior $p(\boldsymbol{\theta})$.
Output: Variational and predictive distributions at each step $\{q_t(\boldsymbol{\theta}), p(y^*|\boldsymbol{x}^*, \mathcal{D}_{1:t})\}_{t=1}^T$.
Initialize the coreset and variational approximation: $C_0 \leftarrow \emptyset$, $\tilde{q}_0 \leftarrow p$.
for $t = 1 \ldots T$ **do**
Observe the next dataset $\mathcal{D}_t$.
$C_t \leftarrow$ update the coreset using C_{t-1} and $\mathcal{D}_t$.
Update the variational distribution for non-coreset data points:

$$\tilde{q}_t(\boldsymbol{\theta}) \leftarrow \arg\min_{q\in\mathcal{Q}} \mathrm{KL}\big(q(\boldsymbol{\theta}) \parallel \tfrac{1}{Z}\tilde{q}_{t-1}(\boldsymbol{\theta})\, p(\mathcal{D}_t \cup C_{t-1} \setminus C_t|\boldsymbol{\theta})\big). \tag{2}$$

Compute the final variational distribution (only used for prediction, and not propagation):

$$q_t(\boldsymbol{\theta}) \leftarrow \arg\min_{q\in\mathcal{Q}} \mathrm{KL}\big(q(\boldsymbol{\theta}) \parallel \tfrac{1}{Z}\tilde{q}_t(\boldsymbol{\theta})p(C_t|\boldsymbol{\theta})\big). \tag{3}$$

Perform prediction at test input $\boldsymbol{x}^*$: $p(y^*|\boldsymbol{x}^*, \mathcal{D}_{1:t}) = \int q_t(\boldsymbol{\theta})p(y^*|\boldsymbol{\theta}, \boldsymbol{x}^*)\mathrm{d}\boldsymbol{\theta}$.
end for

Figure A.4: Pseudo code of the implementation of VCL [56]

A.5. Incremental Moment Matching (IMM)

Incremental Moment Matching (IMM)[35] was published to solve the KL divergence problem between the current parameters distribution and previous pa-

A

rameters distribution by using moment matching in continual learning. The moment matching [57] is usually used to estimate population parameters and deals with distribution approximation problems. When two distributions ($q(\theta|\mu_{T-1}, \sigma_{T-1})$, $q(\theta|\mu_T, \sigma_T)$) are both Gaussian distributions, the expression of KL divergence can be simplified as:

$$KL\left[q(\theta|\mu_{T-1}, \sigma_{T-1})\middle\|q(\theta|\mu_T, \sigma_T)\right] = \frac{1}{2}\left[(\mu_{T-1} - \mu_T)^{\mathrm{T}} \cdot {\sigma_T}^{-1}(\mu_{T-1} - \mu_T) - \log\det({\sigma_T}^{-1}\sigma_{T-1}) + Tr({\sigma_T}^{-1}\sigma_{T-1}) - n\right] \tag{A.14}$$

The first moment of the Gaussian distribution is the average μ, and the second moment is the variance σ. If the first moment of both distributions is equal (μ_T=μ_{T-1}), the KL distance reaches its minimum. In this case, we can infer the first and second moment, which are distribution parameters of $q(\theta|\mu_T)$. This method is Moment Matching[57]. Different from the previous Bayesian-based methods, the update process in IMM is to calculate the distribution of the parameters q_t separately one every single task t and use moment matching to estimate the final distribution $q_{1:T}$ that can remember information in all T tasks based on every previous single distribution q_t. The Incremental Moment Matching is supported by two proposed moment matching algorithms in continual learning scenarios, Mean-based Incremental Moment Matching (mean-IMM) and Model-based Incremental Moment Matching (mode-IMM)[35].

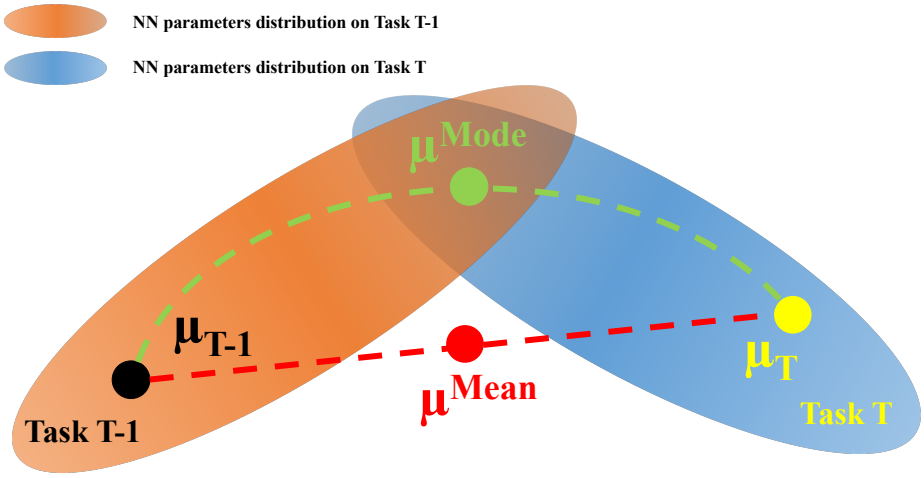


Figure A.5: Schematic diagram for Incremental Moment Matching [35]

In Mean-IMM, the first moment $\mu^*_{1:T}$ of the distribution $q_{1:T}$ is computed by taking averages of the distributions of the parameters on each task by adding a normalization coefficient with $\sum_t^T \alpha_t = 1$[71, 72]. By minimizing the KL divergence,

the parameters of the final distribution $q_{1:T}$ can be obtained by moment matching. In the inference of the parameters of final distribution, only the first moment $\mu_{1:T}^*$, which is also the mean of $q_{1:T}$ is necessary, so the second moment $\sigma_{1:T}^*$ is neglectable. the mean of the final distribution can be computed by the weighted average of every single distribution on each task i, which is shown in Eq.A.15:

$$\begin{aligned}\mu_{1:T}^*, \sigma_{1:T}^* &= \arg\min \sum_t^T \alpha_t KL\left(q_t || q_{1:T}\right) \\ \mu_{1:T}^* &= \sum_t^T \alpha_t \cdot \mu_t \\ \sigma_{1:T}^* &= \sum_t^T \alpha_t \cdot (\sigma_t + (\mu_t - \mu_{1:T}^*) \cdot (\mu_t - \mu_{1:T}^*)^{\mathrm{T}})\end{aligned} \tag{A.15}$$

Mean-IMM avoids complex calculations to transfer the information from each task by using a weighted average of each distribution to solve the parameters of the final distribution. However, the average value of the distributions might not locate in the area in the parameters space that satisfies all low-loss areas for every task. To further limit the update of parameters and force the parameters, mode-IMM is proposed by considering all distributions are mixed with each other to form a new distribution. Since the priori of the distribution is Gaussian distribution, the expression of the final distribution is an MoG (Mixture of Gaussian). The parameters in the T cluster MoG lie on the $T-1$ dimensional hypersurface $\left\{\theta | \theta = \left(\sum_t^T \alpha_t \sigma_t^{-1}\right)^{-1} \cdot \left(\sum_t^T \alpha_t \sigma_t^{-1} \cdot \mu_t\right), with \sum_t^T \alpha_t = 1\right\}$[73]. By using Laplace approximation[74], the parameters of the final distribution are obtained:

$$\begin{aligned}\log q_{1:T} \approx \sum_t^T \alpha_t \log q_t + C' &= \frac{1}{2}\theta^{\mathbf{T}} \cdot \left(\sum_t^T \alpha_t \sigma_t^{-1}\right) \cdot \theta + \left(\sum_t^T \alpha_t \sigma_t^{-1} \cdot \mu_t\right) \cdot \theta + C' \\ \theta = \mu_{1:T}^* &= \sigma_{1:T}^* \cdot \left(\sum_t^T \alpha_t \sigma_t^{-1} \cdot \mu_t\right) \\ \sigma_{1:T}^* &= \left(\sum_t^T \alpha_t \sigma_t^{-1}\right)^{-1}\end{aligned} \tag{A.16}$$

The second moment σ_t is the second derivation of the parameters in the neural network, which introduce a Hessian matrix into the training process. Owing to the computational difficulty of the Hessian matrix, the Fisher information matrix can be applied to approximate the Hessian matrix as the implementation in EWC[24]. In this case, Eq.A.16 can be written as the following version:

A

$$\theta = \mu^*_{1:T} = \sigma^*_{1:T} \cdot \left(\sum_t^T \alpha_t \mathcal{F}_t^{-1} \cdot \mu_t \right)$$
$$\sigma^*_{1:T} = \left(\sum_t^T \alpha_t \mathcal{F}_t^{-1} \right)^{-1} \tag{A.17}$$

B

Appendix to Chapter 4

B.1. Task order for Classification problems

The task order of the MNIST dataset is:

- **task1**: [8, 6]
- **task2**: [9, 0]
- **task3**: [2, 5]
- **task4**: [7, 1]
- **task5**: [4, 3]

Fashion-MNIST Dataset

The task order of the Fashion-MNIST dataset is:

- **task1**: [8, 6]
- **task2**: [9, 0]
- **task3**: [2, 5]
- **task4**: [7, 1]
- **task5**: [4, 3]

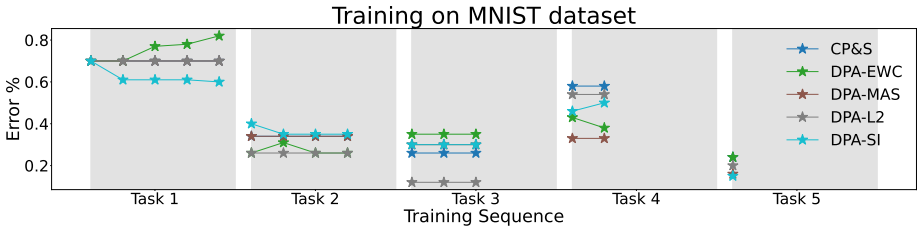
EMNIST Dataset

The task order of the EMNIST dataset is:

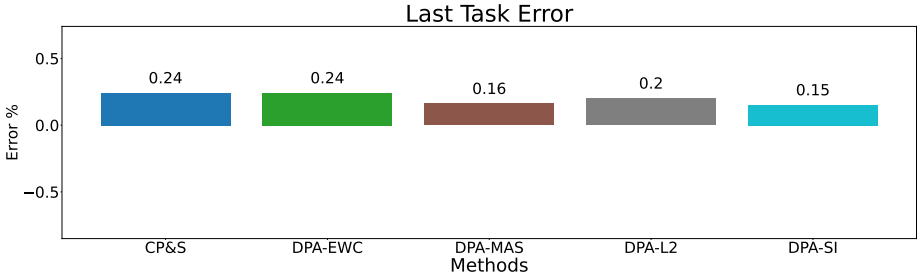
- **task1**: [39, 44, 12, 43, 17]
- **task2**: [23, 32, 0, 31, 22]

- **task3**: [16, 11, 1, 34, 42]
- **task4**: [28, 40, 30, 25, 36]
- **task5**: [7, 37, 4, 38, 33]
- **task6**: [24, 6, 14, 18, 35]
- **task7**: [2, 29, 15, 9, 13]
- **task8**: [27, 10, 21, 19, 8]
- **task9**: [26, 5, 41, 20, 3]

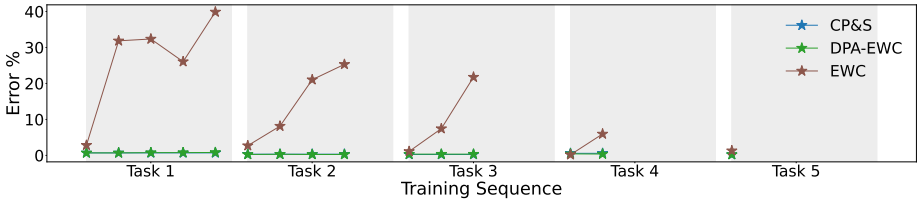
B.2. Results for the MNIST dataset

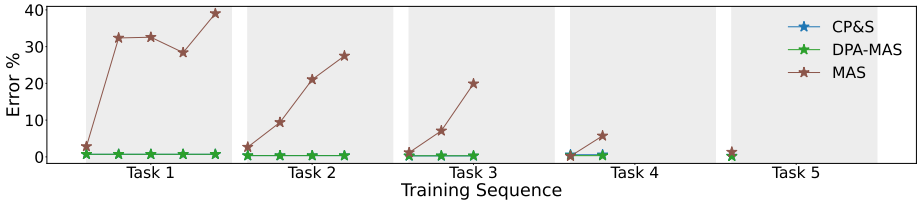


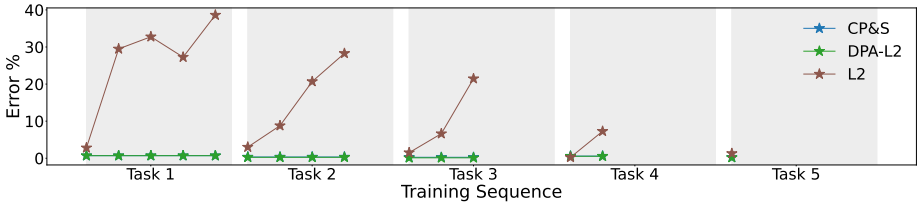
(a) Recognition error for all observation tasks sequence

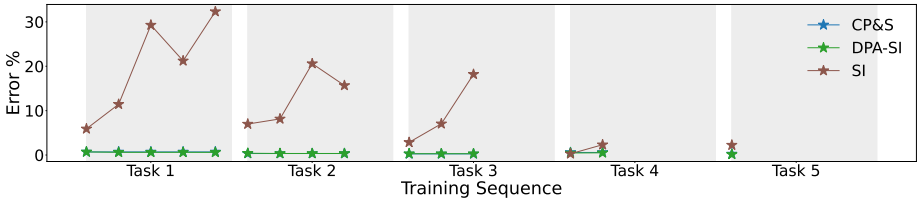


(b) Recognition error after learning the final task





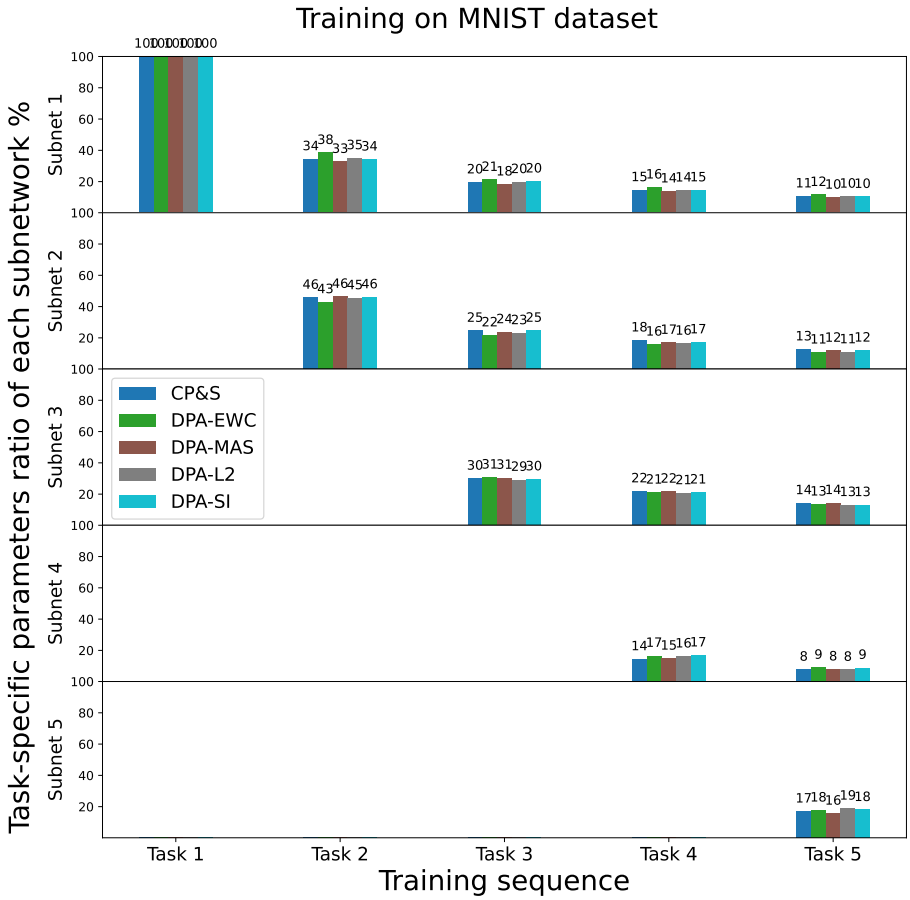




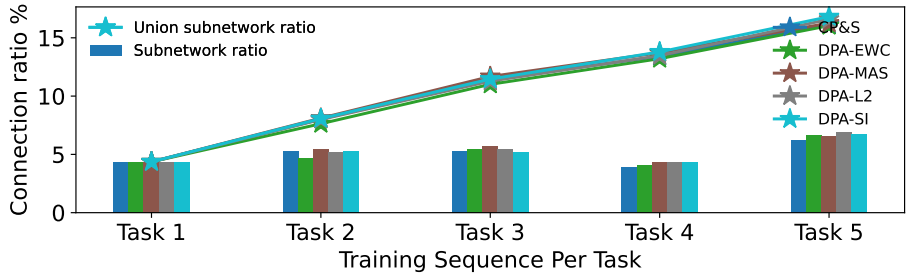
(c) Log recognition error for DPA methods and regularization methods

Figure B.1: The recognition error in the MNIST data experiment

B.3. Sparsity analysis for the experiment in MNIST dataset



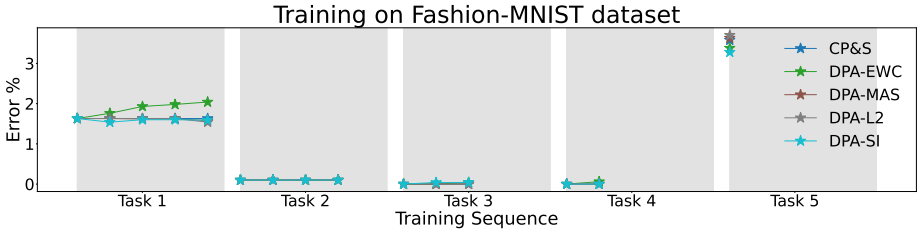
(a) Task-specific parameters ratio for all observation tasks sequence



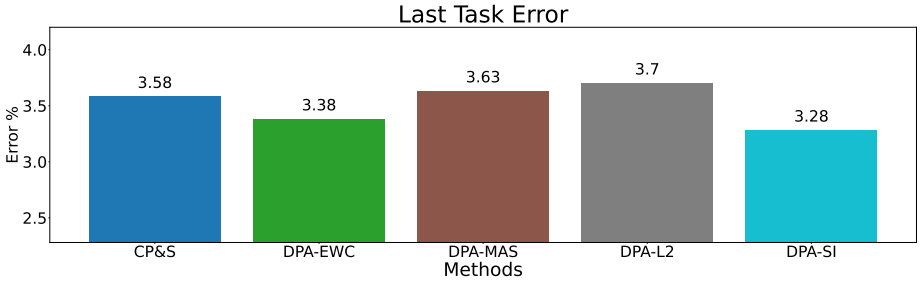
(b) Sparsity analysis for each subnetwork

Figure B.2: Sparsity analysis for the MNIST data experiment

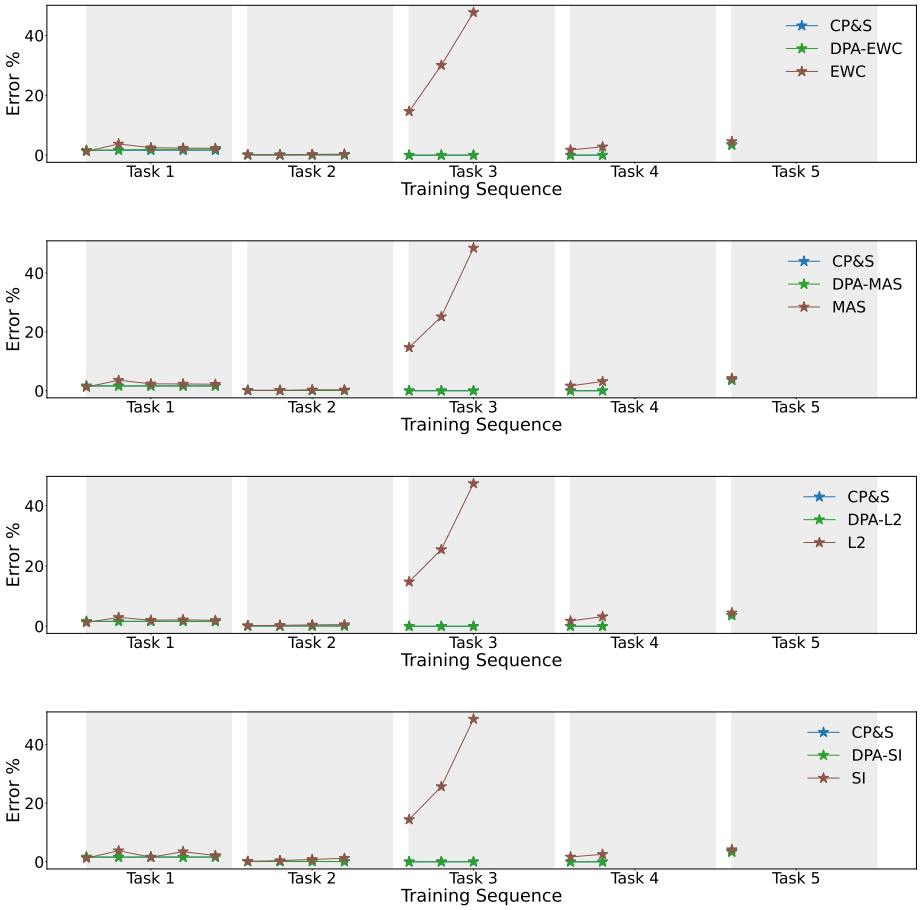
B.4. Results for the experiments in Fashion-MNIST dataset



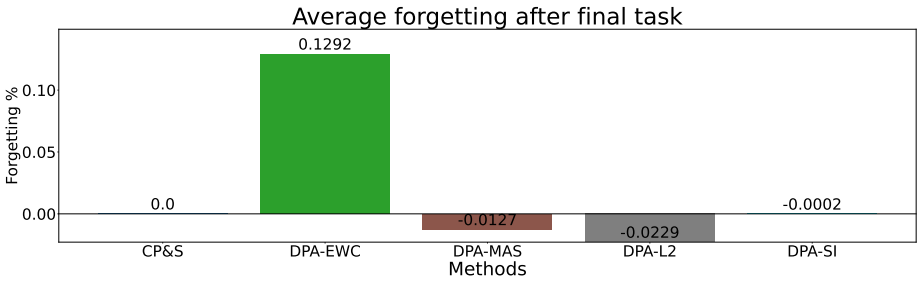
(a) Recognition error for all observation tasks sequence



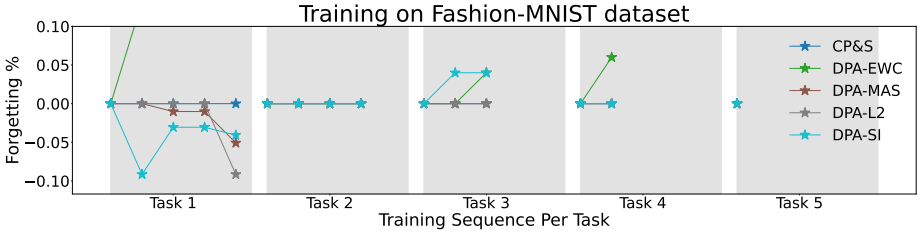
(b) Recognition error after learning the final task



(c) Log recognition error for DPA methods and regularization methods

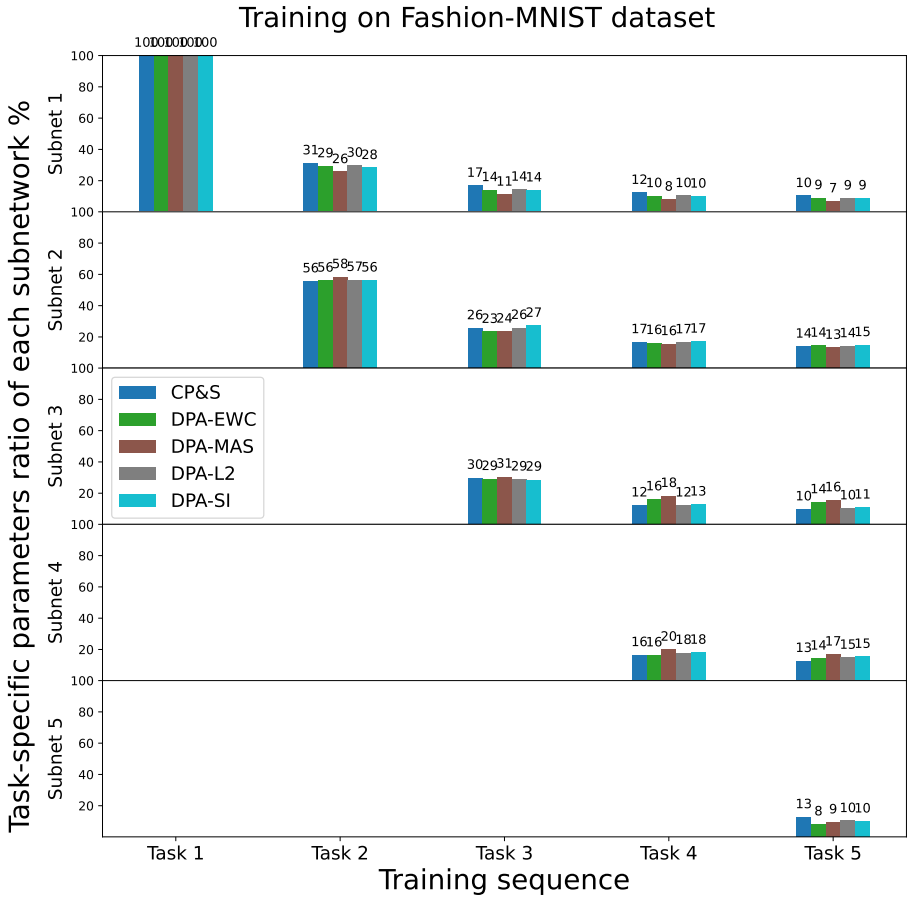


(a) Average forgetting rate after learning the final task

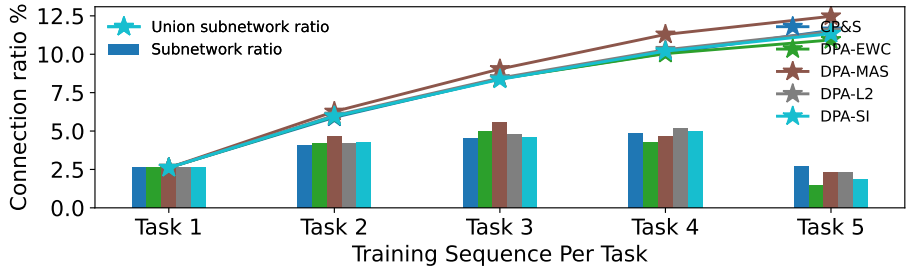


(b) Average forgetting rate for all observation tasks sequence

Figure B.4: The forgetting rate in the Fashion-MNIST data experiment



(a) Task-specific parameters ratio for all observation tasks sequence



(b) Sparcity analysis for each subnetwork

Figure B.5: Sparsity analysis for the Fashion-MNIST data experiment

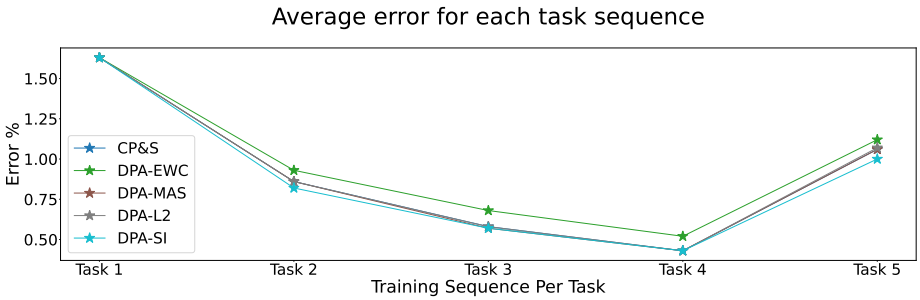
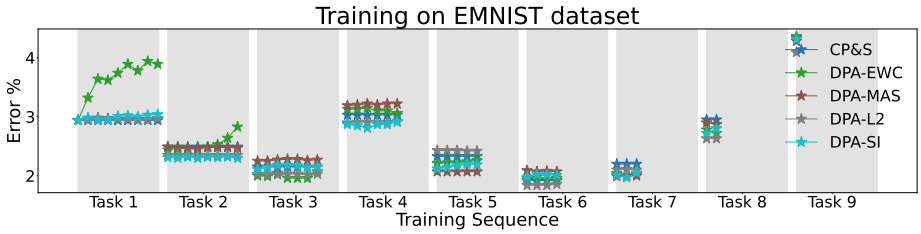
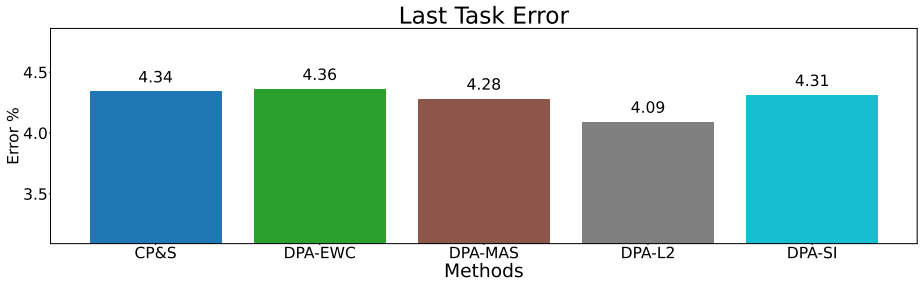


Figure B.6: The average recognition error for all observation tasks in the Fashion-MNIST data experiment

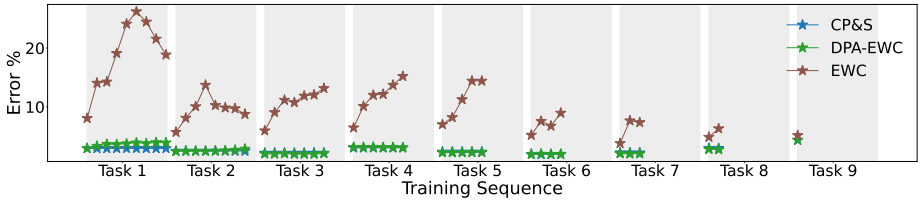
B.5. Results for the experiments in EMNIST dataset

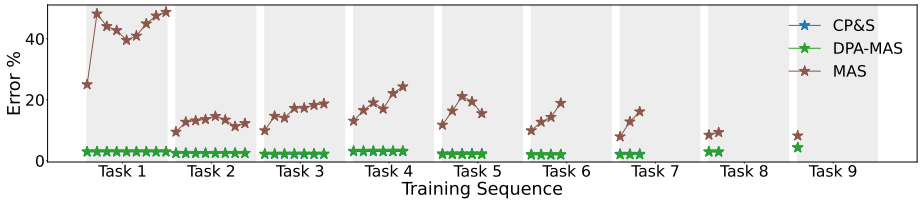


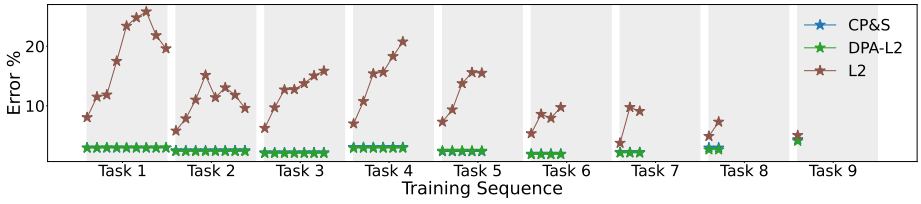
(a) Recognition error for all observation tasks sequence

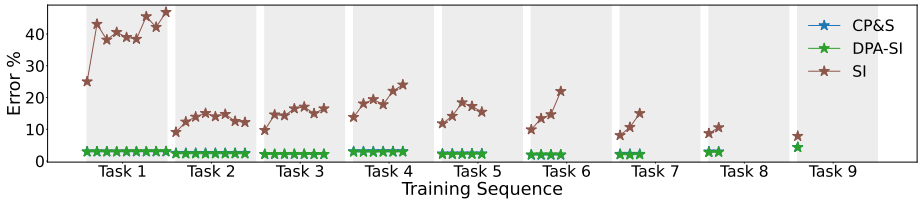


(b) Recognition error after learning the final task



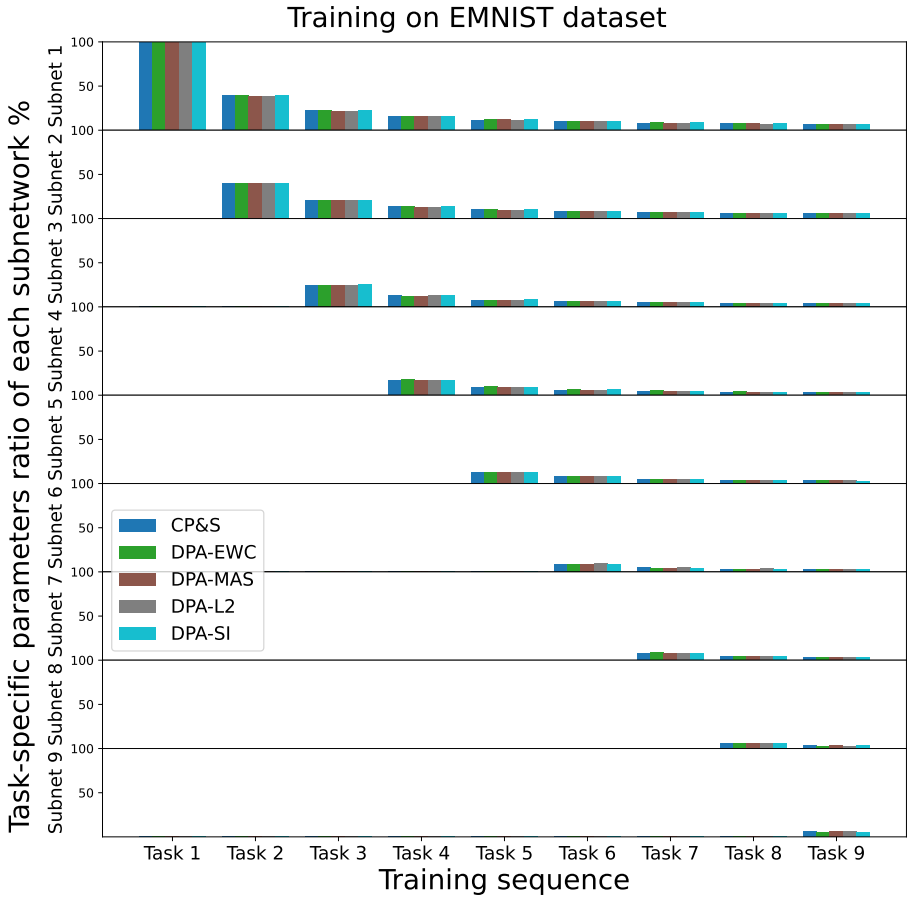




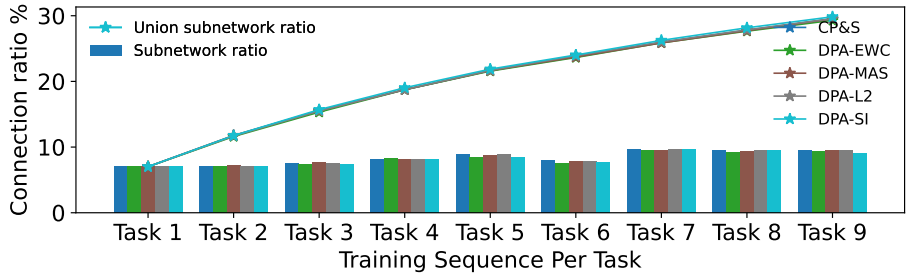


(c) Log recognition error for DPA methods and regularization methods

Figure B.7: The recognition error in the EMNIST dataset experiment



(a) Task-specific parameters ratio for all observation tasks sequence

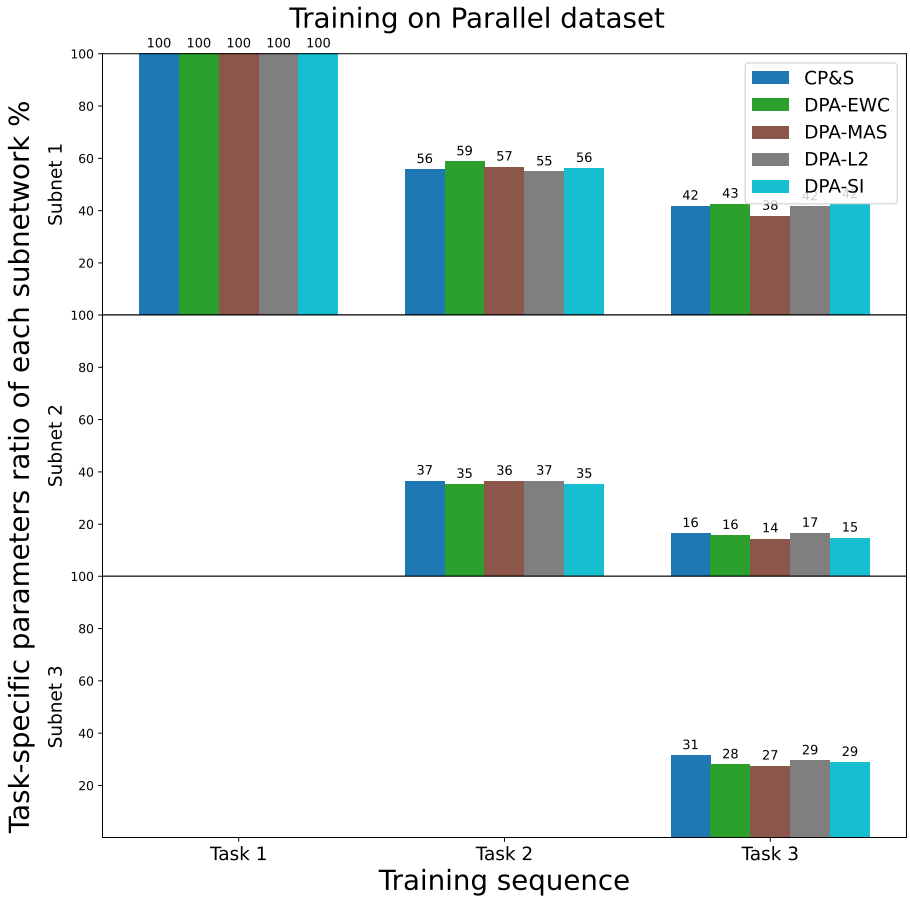


(b) Sparsity analysis for each subnetwork

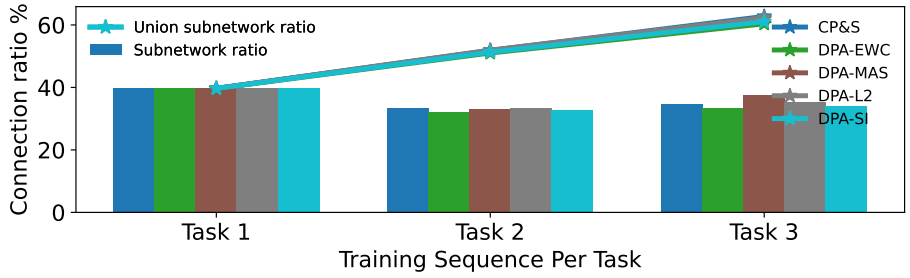
Figure B.8: Sparsity analysis for the EMNIST data experiment

C

Appendix to Chapter 5

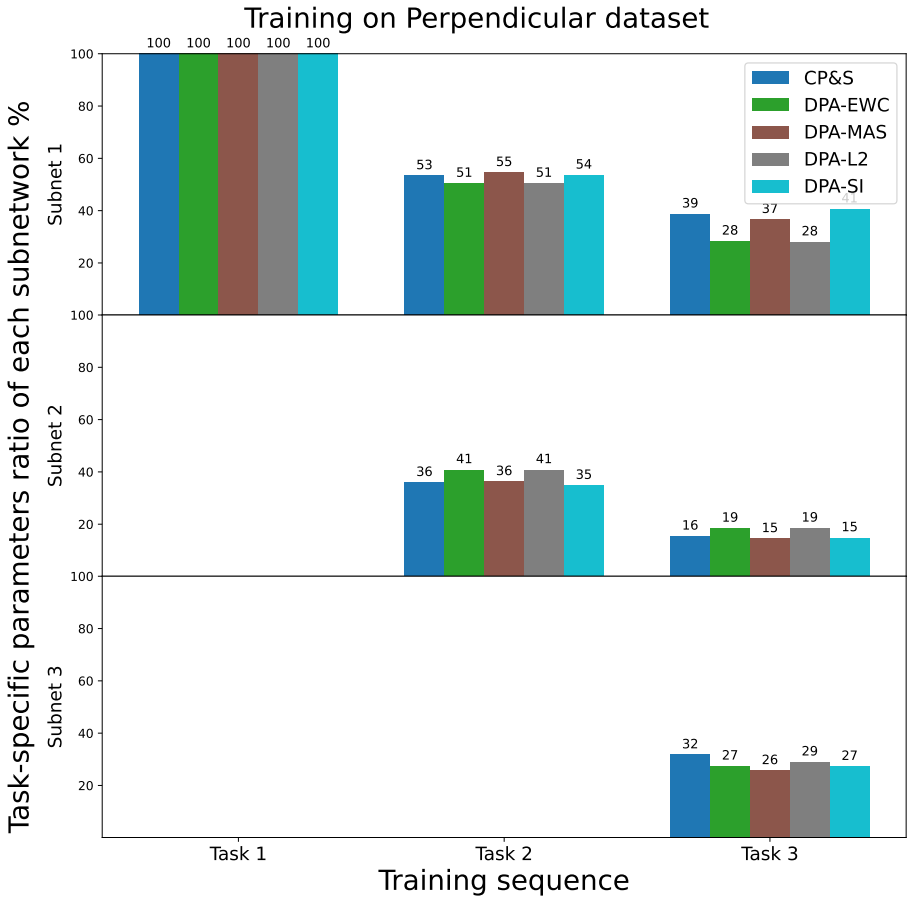


(a) Task-specific parameters ratio for all observation tasks sequence

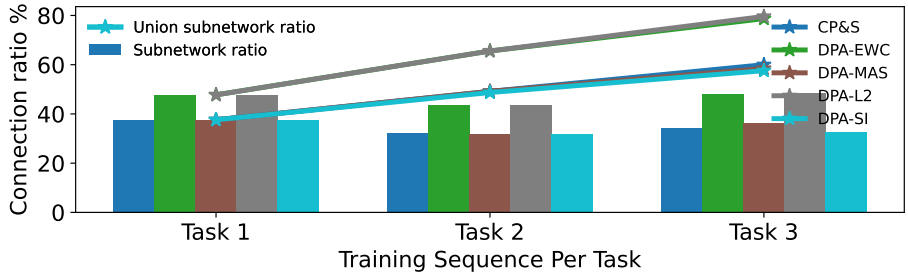


(b) Sparcity analysis for each subnetwork

Figure C.1: Sparcity analysis for the Parallel dataset experiment

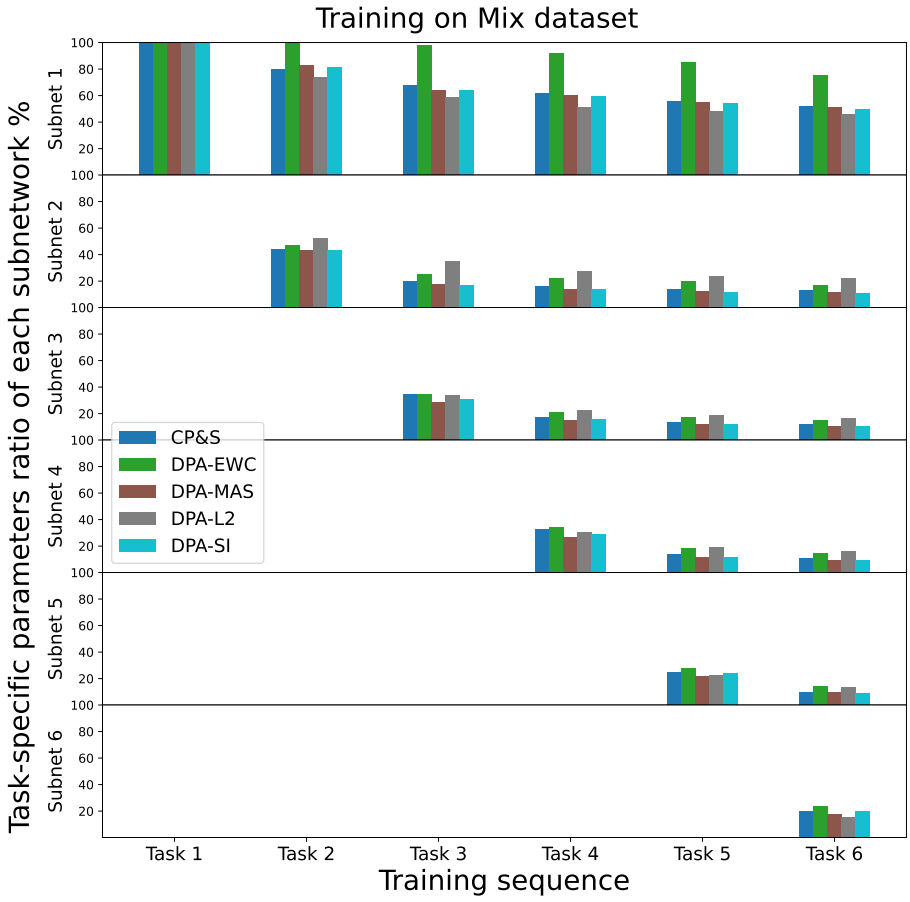


(a) Task-specific parameters ratio for all observation tasks sequence

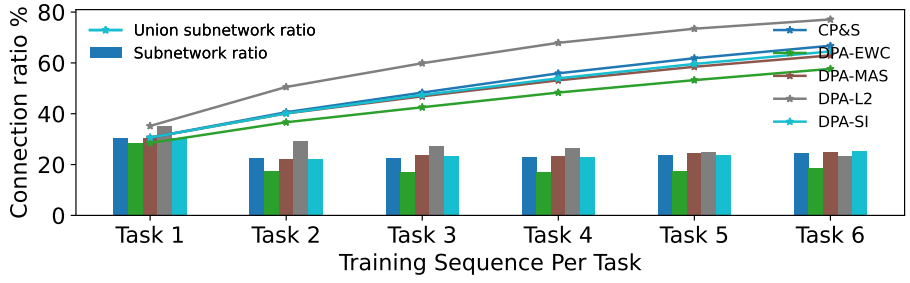


(b) Sparcity analysis for each subnetwork

Figure C.2: Sparcity analysis for the perpendicular dataset experiment



(a) Task-specific parameters ratio for all observation tasks sequence



(b) Sparcity analysis for each subnetwork

Figure C.3: Sparcity analysis for the Mix dataset experiment