Continual Learning Beyond a Single Model

Thang Doan *
Bosch Research North America &
Bosch Center for Artificial Intelligence (BCAI)

Seyed Iman Mirzadeh
Washington State University

Mehrdad Farajtabar†
Apple

Abstract

A growing body of research in continual learning focuses on the catastrophic forgetting problem. While many attempts have been made to alleviate this problem, the majority of the methods assume a *single model* in the continual learning setup. In this work, we question this assumption and show that employing *ensemble models* can be a simple yet effective method to improve continual performance. However, ensembles' training and inference costs can increase significantly as the number of models grows. Motivated by this limitation, we study different ensemble models to understand their benefits and drawbacks in continual learning scenarios. Finally, to overcome the high compute cost of ensembles, we leverage recent advances in neural network subspace to propose a computationally cheap algorithm with similar runtime to a single model yet enjoying the performance benefits of ensembles.

1 Introduction

Continual learning (CL) and Lifelong learning (Thrun, 1994) have recently gained popularity since many real-world applications fall into that setting. It describes the scenario where not only a stream of data arrives sequentially, but their distribution also changes over time. This setup induces Catastrophic Forgetting (CF) (McCloskey & Cohen, 1989) which is a degradation of performances on previous data due to distribution shift between tasks (Doan et al., 2021).

One fundamental goal in continual learning is to learn from the new incoming tasks while retaining knowledge from the past and avoiding interference that can lead to poor performance (Lesort et al., 2021). This becomes particularly challenging when the stream of data increases because all the burden is left to a single model. A simple yet effective solution is to rely on an ensemble method that improves performance over a single model. Inspired by bootstrapping (Breiman, 1996), deep ensembles initialize and train multiple neural networks independently (Lakshminarayanan et al., 2017; Fort et al., 2019). Not only does this improve their robustness, but it also boosts the overall performance thanks to a higher diversity in the solutions and their de-correlated predictions (Goodfellow et al., 2014b; Havasi et al., 2020). This simple mechanism allows ensemble methods to improve performance over single models (Huang et al., 2017). From now on, we will refer to this method as Vanilla Ensemble (VE). It is well known that ensemble methods perform well in supervised learning (Dietterich, 2000; Fort et al., 2019). However, their functionality has not been fully studied in continual learning scenarios.

In the context of continual learning, using an ensemble method and having diversity in the solutions allows each model to have a good solution in different tasks and lead to good performance for the ensemble as shown in Fig. 2. Given that only some of the individual models may need a drastic or a slight change to learn the incoming tasks, it leads to an attenuation of forgetting and a boost in the overall performance (Fig. 1).Recently by Caccia et al. (2022) also showed empirically good performance of ensemble methods in their “anytime learning“ framework where data arrive by batches instead of a whole dataset.

However, the computational cost of ensembles grows linearly with the number of models. This limits their usage for real-world applications due to computational or environmental concerns.

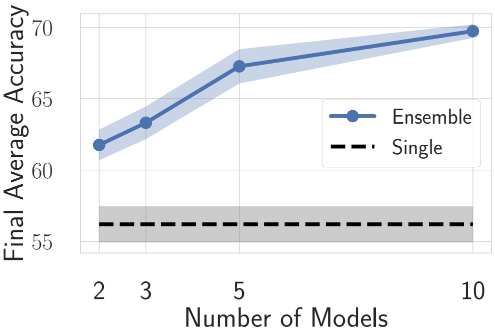


Figure 1: Split CIFAR-100: As the ensemble size grows, the continual learning performance improves.

[*] Work done at McGill University
corresponding author: thang.doan@mail.mcgill.ca
[†] Work done at DeepMind.

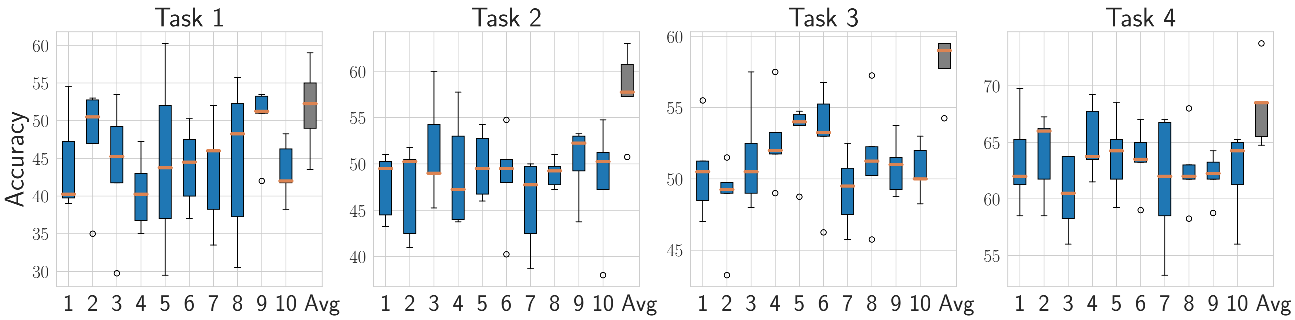


Figure 2: Final accuracy of continual learning after finishing all the tasks for each of the 10 members of the ensemble (blue) and the final ensemble model (gray) for Split CIFAR-100. The final ensemble prediction accuracy is almost always higher than the accuracy of the best model. Moreover, each model specializes in different tasks throughout the training contributing to the performance of the ensemble method.

This issue becomes significantly more challenging on resource-constrained systems such as edge devices (Manolis Savva* et al., 2019; Szot et al., 2021). While it is possible to focus solely on the computational cost problem to find an efficient parameter sharing algorithm (Wen et al., 2020), in this work, we focus on both the computational efficiency and continual learning performance together. Our work is motivated by the recent advances in the deep learning optimization literature, such as Neural Network Subspace (Wortsman et al., 2021) and Mode Connectivity (Garipov et al., 2018; Draxler et al., 2018; Mirzadeh et al., 2021).

Contributions. In this work, we study continual learning with various ensemble methods. Our contributions can be summarized as three-fold: we show a) that ensembling is a simple technique to boost continual learning performance, but significantly increasing the computation cost. To alleviate this issue, we borrow insights from the recent advances mode-connectivity to propose b) a method with a similar computation cost to a single model yet enjoys the high-performance benefits of ensembles. Our proposed efficient ensembling method learns tasks continually in the subpaces of neural networks (Wortsman et al., 2021) to divide the learning capacity between tasks without causing interference, and relies on the connectivity of the tasks' optima (Garipov et al., 2018) for better retention of the previous knowledge.

2 Preliminaries

Notation. Let $\mathcal{X}$ be some features and $\mathcal{Y}$ the labels space ($\mathcal{Y} = \mathbb{R}$ for a regression problem and $\mathcal{Y} \in \Delta^K$ for classification problem[1]). In CL, a stream of supervised learning tasks indexed by $\tau \in [T]$, $\mathcal{T}_\tau$, (where $T \in \mathbb{N}^*$ is the total number of tasks) arrives sequentially. The goal is to learn a predictor $f_\omega : \mathcal{X} \times \mathcal{T} \to \mathcal{Y}$ (where $\omega \in \mathbb{R}^p$ are the learnable parameters of size p) that performs a prediction as accurate as possible with respect to a loss function $\mathcal{L}_\tau$. We denote the weight learned after task τ as ω_τ^*. In the framework of CL, one cannot recover samples from previous tasks unless storing them in a replay buffer (Chaudhry et al., 2019b). A pseudo-code of the Vanilla Continual Learning Algorithm (single model) is provided in Appendix (Alg. 2).

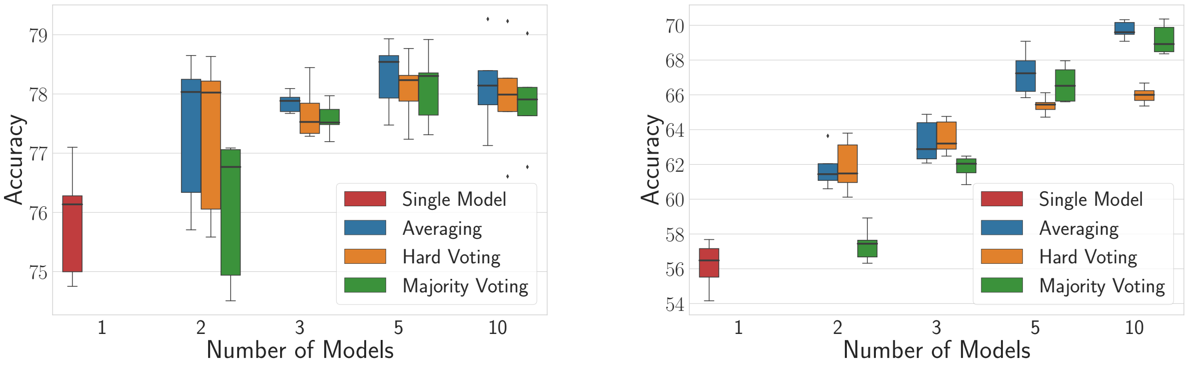
2.1 Experimental Setup

For our experiments in Sec. 3, we will be using the following setup:
Benchmarks. For this ablation, we use two standard benchmarks following (Goodfellow et al., 2014a), and (Chaudhry et al., 2019b). Rotated MNIST (Farajtabar et al., 2020) consists of a series of MNIST classification tasks, where the images are rotated with respect to a fixed angle, monotonically. We increment the rotation angle by 22 degrees at each new task like in Mirzadeh et al. (2021) to worsen the catastrophic forgetting phenomenon. The task ID does not need to be provided since it is a domain incremental task. Split CIFAR-100 (Chaudhry et al., 2019b) is constructed by splitting the original CIFAR-100 dataset (Krizhevsky et al., 2009) into 20 disjoint subsets, where each subset is formed by sampling without replacement of 5 classes out of 100. For the sake of our experiment, we fine-tune on a sequence of 5 tasks for each benchmark, seeing the tasks only once without relying on a replay buffer. For this dataset, the task ID is provided to the model. While for brevity, we include the main results in this section, more detailed plots can be found in Appendix D.
Architectures. The neural network architectures used are respectively fully connected layers with two hidden layers of

[1] Δ^K denotes the vertices of the K-dim probability simplex

100 hidden units (Rotated MNIST) and a reduced Resnet-18 with three times fewer filters map across all layers as in Mirzadeh et al. (2020b).

Metrics. To assess the performance of each baseline, we report the Final Accuracy and Forgetting Measure defined as follows. The Final Accuracy after T tasks is the average validation accuracy over all the tasks $\tau = 1...T$ defined as: $A_T = \frac{1}{T}\sum_{\tau=1}^{T} a_{T,\tau}$ where $a_{T,\tau}$ is the validation accuracy of task τ after the model finished learning on task T (at test time). The Learning Accuracy is defined as $a_{\tau,\tau}$, this describes how well a model learns a task τ the first time it sees it. The Forgetting Measure is defined as: $F_T = \frac{1}{T-1}\sum_{\tau=1}^{T-1} \max_{t=\{1..T-1\}}(a_{t,\tau} - a_{T,\tau})$. Finally, we define the Forgetting Improvement (FI) simply as the difference between the Forgetting Measure of the single model and the ensemble (or subspace) method as: $FI_T = F_T(\textit{single model}) - F_T(\textit{ensemble/subspace model})$. Intuitively, the higher this value the less forgetting a method has compared to the single model case.



(a) Rotated MNIST

(b) Split CIFAR-100

Figure 3: Performance of different prediction strategies for Vanilla Ensemble over 5 seeds for $n = 1, 2, 3, 5$ and 10 models (x axis). Averaging strategy often outperforms the other strategies.

3 Continual Learning With Ensembles

This section studies the implications of having multiple models for continual learning. We start by comparing basic ensembling strategies and then we study more efficient ensembling techniques. For brevity, we postpone detailed discussion and additional results to Appendix A and D.

Algorithms. For this analysis, we compare Vanilla CL (Alg. 2), Ensemble CL (Alg. 3), Batch Ensemble CL and Subspace CL (Alg. 4).

3.1 Does the Ensembling Strategy Matter?

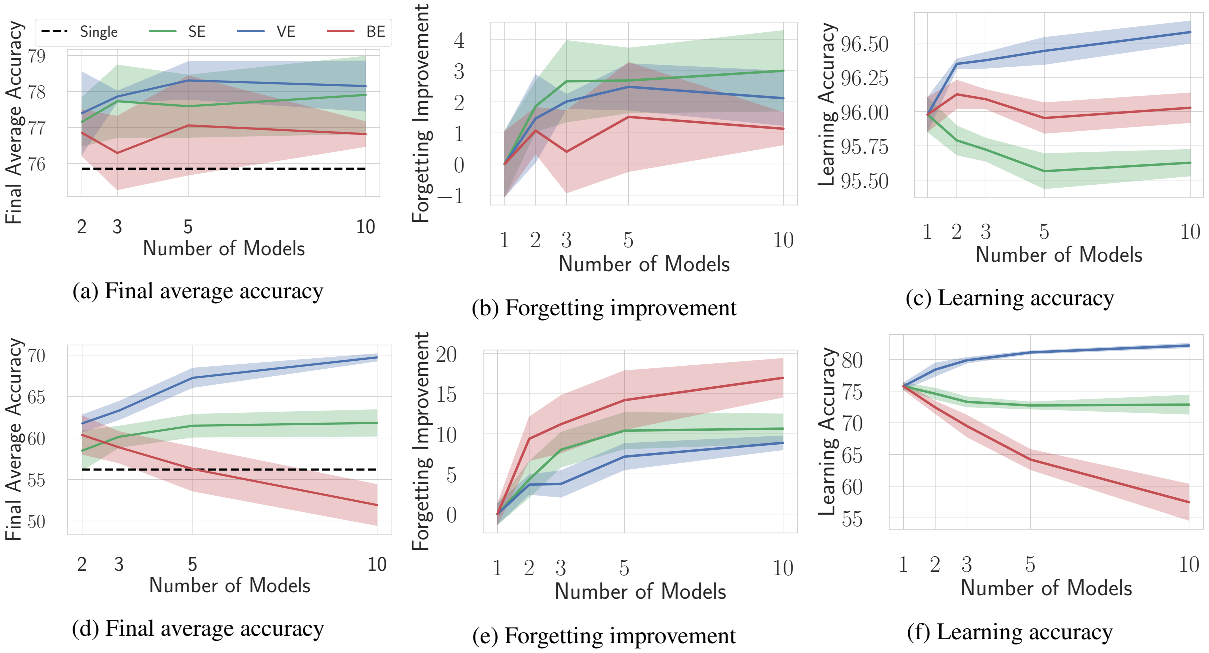
Ensembles can use various strategies for learning and prediction. Here, we study three common choices:

- **Averaging**: Average of predictions for each ensemble member.
- **Hard Voting**: Selecting the label for which a member is the most confident.
- **Majority Voting**: Label that gathers the most vote among the members.

As shown in Fig. 3, averaging strategy consistently outperforms (slightly) other strategies on both benchmarks. In non continual learning scenarios this is consistent with previous findings on the benefits of averaging mechanism (Caccia et al., 2022; Huang et al., 2017; Caruana et al., 2004). In the context of continual learning, we hypothesize that averaging considers each member's diversity, which is a good fit for multiple tasks and distribution and can boost performance in continual learning. Figure 9 provides a visualization of Vanilla ensemble's prediction evolution throughout the learning highlighting the diversity in solution.

3.2 Different Ensembling Methods

Although Vanilla Ensemble (VE)[2] shows promising performance benefits, its computation cost grows linearly with the ensemble size. Both Batch Ensemble (BE (Wen et al., 2020)) and Subspace Ensemble (SE (Wortsman et al., 2021)) provide more efficient ensembling techniques. Hence, for the rest of this section, we study all three methods from performance and computation perspectives.



(a) Final average accuracy

(b) Forgetting improvement

(c) Learning accuracy

(d) Final average accuracy

(e) Forgetting improvement

(f) Learning accuracy

Figure 4: Continual Learning metrics for Rotated MNIST (first row) and Split CIFAR-100 (second row). Having more models improve the final average accuracy over a single model (**left**). While SE and BE methods mitigate forgetting (stability) by tying the weights of its member (**middle**), the VE method enjoys better learning accuracy (plasticity) as each model moves independently (**right**).

Vanilla Ensemble (VE) Given a set of weights $\{\omega_i\}_{i=1}^n$, we train independently n models by optimizing $\frac{1}{n}\sum_{i=1}^n \mathcal{L}_\tau(f_{\omega_i}(x), y)$. Therefore, the models' training differs only in the weights initialization of each model. For the prediction we use the average of each member $\frac{1}{n}\sum_{i=1}^n f_{\omega_i}(x)$.

Batch Ensemble (BE) Let's consider a neural network layer weights $W \in \mathbb{R}^{k\times l}$ where k and l are the input and output dimension. BE factorizes the weights ω such that each member of the ensemble i has weights $\omega_i = \omega \circ f_i$ with $f_i = r_i s_i^T$, $r_i \in \mathbb{R}^k, s_i \in \mathbb{R}^l$. In other words, while they share a common weights ω they have their own tuple $\{r_i, s_i\}$. During training, each element of the incoming batch (x, y) is shared uniformly among the member of the ensemble. For the prediction we use the ensemble's average prediction as suggested in (Wen et al., 2020).

Subspace Ensemble (SE) Given a set of weights $\{\omega_i\}_{i=1}^n$, SE trains a predictor $f_{\hat{\omega}}$ such that $\hat{\omega} = \sum_{i=1}^n \alpha_i \omega_i$ with $\sum_{i=1}^n \alpha_i = 1$, i.e forming a convex combination of the weight of each member. For the prediction we use the midpoint defined as $\frac{1}{N}\sum \omega_i$.

We refer the reader to Appendix A for more details about each of the ensemble methods as well as their pseudo-code in Appendix B.

[2] For the rest of the paper, unless otherwise stated, VE employs the averaging strategy.

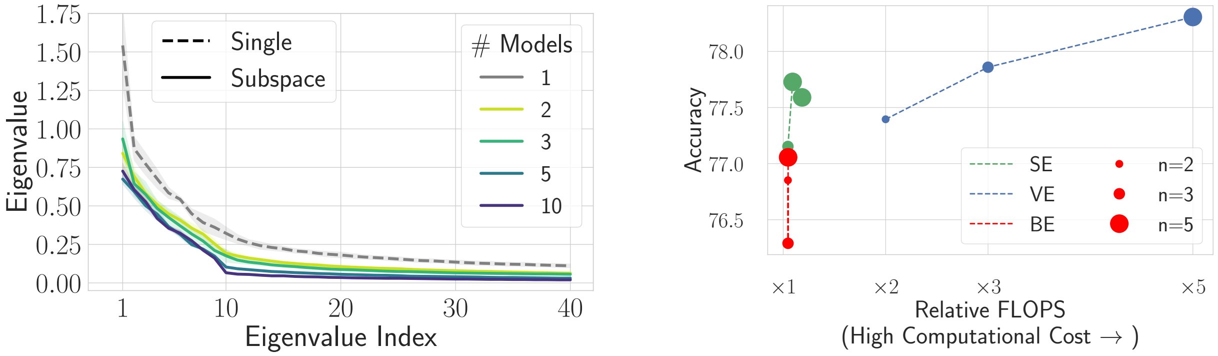


Figure 5: **(left)** The Hessian spectrum of each model for the loss of task 1 for Rotated MNIST. For SE methods, the more models, the lower the eigenvalues. Lower eigenvalues imply flatter minima which is a proxy to describe how much forgetting will be incurred when learning subsequent tasks. **(right)** Accuracy versus cost in x axis (wrt to single model) for each Ensemble method for Rotated MNIST on 5 tasks. Although VE shows better performance, it also implies linear growth in cost, unlike SE and BE, which have close to single model compute cost.

3.2.1 Performance Comparison

We first focus on the continual learning performance of each of the aforementioned methods. More specifically, we focus on the average accuracy, forgetting improvement, and learning accuracy that provide interesting insights on the learning and retention capabilities of each ensembling method.

Average Accuracy. Fig. 4a and 4d show the final average accuracy on both Rotated MNIST and Split CIFAR-100 benchmarks for all three methods. We can see the immediate benefits of both ensemble methods compared to the single model (black dashed line).

Average Forgetting. To better understand these results, we show the *forgetting improvement* (FI) which describes how much a model forgets compared to a single model. Overall, all ensembling methods provide a better forgetting improvement compared to a single model (Fig. 4b and 4e).

Learning Accuracy. Figs. 4c and 4f show the learning accuracy for each method which demonstrate their ability to learn new tasks. Not surprisingly, ensemble methods with the best forgetting improvement (SE and BE) trade their stability for less plasticity since the models in the ensemble are tightly tied together. On the other hand, since VE trains each of its members independently, it enjoys a better learning accuracy and a better final accuracy.

SE mitigates CF by tying the weight of each member (through the weight's convex combination). This is illustrated by the eigenvalues of the Hessian of the loss function (Fig. 5). On the other hand, VE trains its member independently. We believe the power of ensembles and their diversity-enhancing approach (due to random initialization) is very pronounced in the context of continual learning. For BE, we adapted the algorithm from Wen et al. (2020) where originally one member is added for each new task (with a total of 250 epochs) to our minimal setting of 1 epoch per task where members are responsible for all tasks similarly to VE. Since the batch is divided equally among all members, this looks similar to bagging strategy and finally, each member only gets to see $1/n$ of the whole dataset.

Given the good performance of ensembles, one may keep adding more models for continued improvement. However, it is either not always helpful (MNIST dataset) or the marginal improvement is not worth the linear compute cost. See Figs 4d and 4e for the relevant results. For the subspace method, increasing the number of models (hence the number of parameters) requires more training epochs to update them. That can explain why after a certain threshold (e.g., $n = 5$ models for Split CIFAR-100), the accuracy and forgetting metrics degrade or plateau given a fixed training budget. We now elaborate on the compute cost implied by these additions of models.

3.3 Performance-Cost Trade-off

Another critical factor for comparing ensembles is their compute cost. For convenience, we will refer to the (Inference) FLOPS as the compute cost (The backward pass can be approximated as two times the forward pass). The relative FLOPS is the compute cost ratio wrt to the single model. Table 1 provides the compute cost implication of each ensemble method for a single layer as a proof of concept. VE requires the most compute cost with linear growth while SE and BE have an addition and Hadamard product as overhead costs. Fig. 5 (right) shows the accuracy-cost trade-off for VE, SE, and BE. We can observe that VE is the least cost-efficient method while SE and BE have roughly similar costs to a single model. However, SE enjoys higher accuracy compared to BE.

Method	Inference Step	Overhead Cost
Single Model	$x \cdot \omega$	N/A
Vanilla Ens.	$x \cdot \omega_i, \forall i = 1..n$	$(n-1)$ forward pass
Subspace Ens.	$x \cdot (\sum_{i=1}^{n} \alpha_i \omega_i)$	$(n-1)$ additions
Batch Ens.	$((x \circ R) \cdot W) \circ S$	2 Hadamard products

Table 1: Computational cost comparison between different methods for a given layer ω and input x. SE only has addition operations as an overhead cost, while BE has Hadamard products independently of the size of the ensemble.The first line corresponds to the single model as a reference.

Overall, each of the above ensembling methods has its own mechanism to mitigate CF (either with diversity enhanced, tying its member's weights together). However, VE is not computationally efficient, and BE does not provide strong performance. For these reasons, we focus in the next section on improving Subspace Ensemble to provide an efficient alternative approach to the high compute cost of Vanilla Ensemble.

4 Improving Subspace Learning with Connectivity

In Sec.3, we observed that although SE enjoys a nice compute cost, it cannot match the performance of the VE method. In this section, we aim to find out if we can improve the performance of the SE method in continual learning scenarios while keeping the cost roughly the same.

To improve the performance, Experience Replay (ER) Riemer et al. (2018) is one of the most popular and practical CL methods that come to mind at first. However, naively adding ER method to subspace will not increase the performance significantly. To explain, we note that the subspace formed by models of the SE method is subject to drift as the models' parameters still change as the task changes. As a result, the optimal subspace found by the SE method will drift as the number of tasks increases. Appendix D.5 provides a more detailed analysis of this subject.

To prevent the subspace drift problem and thus exploit the benefits of the ER method, we note that SE method Wortsman et al. (2021) is originally motivated by mode connectivity of optima Garipov et al. (2018). In the context of continual learning, Mirzadeh et al. (2021) has shown that enforcing the linear mode connectivity of each task's optima is equivalent to mimicking the multitask (i.e., joint) training and hence, a key factor in preventing forgetting. Given the shared origins between Wortsman et al. (2021) and Mirzadeh et al. (2021), it is natural to think about both works together to overcome the drift challenge. However, Mirzadeh et al. (2021) studies continual learning with a "single" model, while here, we work with "multiple" models that form a convex region. Intuitively, our algorithm is the generalization of the proposed algorithm by Mirzadeh et al. (2021) where we prevent the drift of an optimal subspace rather than a single optimal (i.e., model).

Our Subspace-Connectivity algorithm proceeds in two steps: we first naively finetune to the incoming task to learn a solution ω^*_τ before creating a low-loss path to the former task's solutions $\omega^*_{\tau-1}$ that enforces the connectivity between subspaces found for each task. A pseudo-code can be found in Alg. 1.

(1) Learning a subspace solution for the incoming task: The first step consists in learning a subspace solution by fine-tuning on task τ leading to the solution $\hat{W}_\tau = \{\hat{\omega}_{\tau,i}\}_{i=1}^{n}$ obtained by optimizing:

$$\{\hat{\omega}_{\tau,i}\}_{i=1}^{n} = \underset{W}{\text{argmin}} \quad \mathbb{E}_{\boldsymbol{\alpha} \sim \mathcal{U}[\Delta^n]}[\mathcal{L}_\tau(W^T \boldsymbol{\alpha})] \tag{1}$$

At the end of this step, we save in a buffer memory $\mathcal{B}$, $m_{\mathcal{B}}$ samples per class per task that will be used to connect linearly two subspace's solutions.

(2) Connecting the new subspace to previous subspaces: This step aims at connecting subspaces from prior task's solutions together as in MC-SGD Mirzadeh et al. (2021). First, we use the midpoint of subspace τ denoted $\hat{\omega}^*_{\tau,mid}$ as its proxy since it gives the best performance (Sec. D.6). We then connect $\hat{\omega}^*_{\tau,mid}$ and previous tasks midpoint solution $\omega^*_{\tau-1,mid}$ via a low loss path. The loss over the connecting path acts as a penalty or regularizer term. The rationale behind choosing the middle point as a proxy for subspace is that the middle point is the most stable point of the subspace (Wortsman et al., 2021). In our experiments in the appendix, we also confirm this observation in the context of continual learning.

Algorithm 1 Subspace-Connectivity CL

```
Input  : A task sequence $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_T$, number of models $n$ , buffer $\mathcal{B}$ and memory size $m_\mathcal{B}$
Initialize set of weigths $S^n = \{\omega_{0,i}\}_{i=1}^n$, buffer $\mathcal{B} \leftarrow \{\}$
 for tasks $\tau = 1, 2, 3, \ldots T$ do
    Get $\{\hat{\omega}^*_{\tau,i}\}_{i=1}^n$ with Eq 1   // Learn subspaces solution for task $\tau$
    for $(x, y) \in \mathcal{T}_\tau$ do
        $\mathcal{B} \leftarrow \mathcal{B} \bigcup \{x, y\}$ // Collect 1 sample per class at the end of each class
    end for
    Get $\{\omega^*_{\tau,i}\}_{i=1}^n$ with Eq 2 using samples from $\mathcal{B}$   // Connect previous solution' subspaces
end for
Output : $\{\omega^*_{T,i}\}_{i=1}^n$
```

Thus, for the second step of our algorithm, we optimize the following objective with saved elements from the buffer:

$$\{\omega^*_{\tau,i}\}_{i=1}^n = \underset{W}{\text{argmin}} \quad \mathbb{E}_{\boldsymbol{\alpha}\sim\mathcal{U}(\Delta^{n+1})}[\sum_{j=1}^{\tau-1}\mathcal{L}_j(W^T\boldsymbol{\alpha}^n + \quad \alpha_{n+1}\omega^*_{\tau-1,mid})] + \mathcal{L}_\tau(W^T\boldsymbol{\alpha}^n + \alpha_{n+1}\hat{\omega}_{\tau,mid})] \qquad (2)$$

where $\boldsymbol{\alpha} = (\underbrace{\alpha_1, .., \alpha_n}_{\boldsymbol{\alpha}^n \in \mathbb{R}^n}, \alpha_{n+1}) \in \mathbb{R}^{n+1}$ and $\mathcal{L}_j$ corresponds to the loss of task j (using element of task j from the buffer). The rationale behind this loss function is to create a linear path of low-loss between two subsequent solutions $\omega^*_{\tau-1,mid}$ and $\omega^*_{\tau,mid}$.

A few remarks worth noting here. Intuitively speaking, the subspace method ties the models for a single task, while the mode connectivity regularization ties the Subspaces together. Note that the original subspace method is only developed for single-task settings. Although SE is an efficient ensembling technique, it still does not have any mechanism to learn from a sequence of tasks.

5 Experiments and Results

In this section, before comparing Subspace-Connectivity to Ensemble baselines, we want to address a subsidiary yet important question: "Will increasing the number of parameters (for instance, the number of hidden units) boost performance, or is it the way parameters work together as an ensemble that matters?" To address this hypothesis, we compare the Ensemble model with a Scaled version of a single model where we either increase the number of hidden units (for dense networks) or the number of filters (for convolutional networks) to match the number of parameters in Ensemble models. Finally, we compare our algorithm with an Ensemble version of MC-SGD (Mirzadeh et al., 2021) (dubbed Ens MC-SGD) and provide a discussion between the trade-off accuracy and compute cost. As a note, we use MC-SGD for the experiments in this section since it is one of the strongest CL methods and shares a similar mechanism to our proposed method, so the comparison is fairer. Although the main focus of our work is ensemble methods, we additionally report the results for other ensemble methods (e.g., Batch Ensemble) and also single models with various learning algorithms (e.g., A-GEM, ER, etc.) in Appendix F. Since we want to simulate an online setting, which is relevant for small AI embedded devices, we ran all algorithms with only 1 epoch per task.

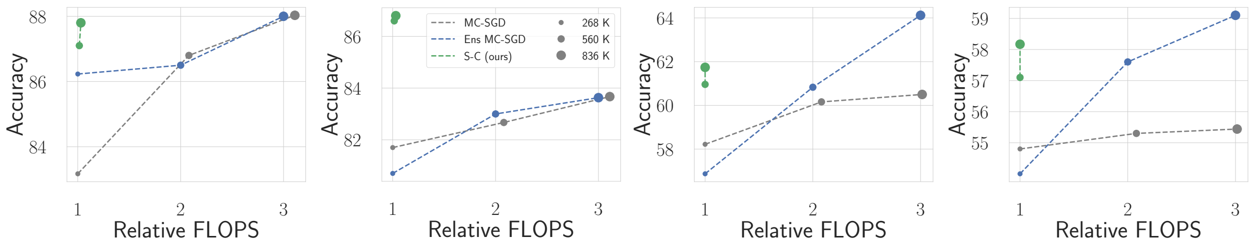
5.1 Does Increasing the Number of Parameters Help?

We first compare Ensemble MC-SGD (with $n = 3$) to Scaled MC-SGD, where the number of hidden units has been increased to 600 and the number of kernel filters to 35, to match the parameters and compute cost. Table 2 showcase the performance of both models. We can observe that across all benchmarks, even with the same number of parameters and thus computation, the Ensemble MC-SGD model matches or outperforms the Scaled MC-SGD. Similar to our discussion in Sec. 1, we can see that the benefit of ensembles is not just the increase of parameters but the way the models communicate with each other, which can lead to more diverse solutions.

Method	**Permuted MNIST**		**Rotated MNIST**		Relative FLOPS ratio
	Accuracy ↑	Forgetting ↓	Accuracy ↑	Forgetting ↓	
MC-SGD (Mirzadeh et. al Mirzadeh et al. (2021))	82.9 (±0.40)	0.10 (±0.01)	81.9 (±0.46)	0.08 (±0.01)	1
Scaled MC-SGD	88.03 (±0.36)	0.06 (±0.01)	83.67 (±0.40)	0.07 (±0.01)	3.11
Ensemble MC-SGD	88.30 (±0.48)	0.06 (±0.01)	83.63 (±0.39)	0.07 (±0.01)	3
Subspace-Connectivity	87.8 (±0.30)	0.07 (±0.01)	86.7 (±0.67)	0.07 (±0.01)	1.03
Multitask Learning	89.5 (±0.21)	0.0	89.8(±0.37)	0.0	NA

Method	**Split CIFAR-100**		**Split miniImageNet**		Relative FLOPS ratio
	Accuracy ↑	Forgetting ↓	Accuracy ↑	Forgetting ↓	
MC-SGD (Mirzadeh et. al Mirzadeh et al. (2021))	58.22 (±0.91)	0.08 (±0.01)	54.80 (±1.04)	0.03 (±0.01)	1
Scaled MC-SGD	60.55 (±0.84)	0.06 (±0.01)	55.44 (±1.36)	0.05 (±0.01)	3.01
Ensemble MC-SGD	64.12 (±1.16)	0.06 (±0.01)	59.10 (±1.1)	0.04 (±0.01)	3
Subspace-Connectivity (ours)	61.7 (±0.80)	0.05 (±0.01)	58.17 (±0.84)	0.03 (±0.01)	≈ 1
Multitask Learning	66.8 (±1.42)	0.0	62.82(±1.77)	0.0	NA

Table 2: Comparison ensemble methods performance with $n = 3$ models or the equivalent number of parameters for Scaled MC-SGD. Each benchmark includes 20 tasks.



(a) Permuted MNIST

(b) Rotated MNIST

(c) Split CIFAR-100

(d) Split miniImageNet

Figure 6: Performance of (Scaled) MC-SGD, Ensemble MC-SGD (Ensemble) and Subspace-Connectivity (S-C) with respect to the inference cost (FLOPS) and number of parameters/models n (circles) on 20 tasks.

5.2 Performance Vs. Compute Trade-off

Fig. 6 reports the performance versus the relative FLOPS (w.r.t to MC-SGD). We notice that for fully connected models (MNIST) Subspace-Connectivity is at least on par with Ensemble MC-SGD and Scaled MC-SGD but enjoys a much better compute cost (close to Single MC-SGD). However, for convolutional models, Ensemble MC-SGD has the best performance but at a high computational cost. One reason might be due to the small dataset size (500 per task) on which Subspace models are not performing well. Note that the subspace method incurs almost no computational cost overhead in this case. This can be explained by the high reuse of kernel filters in convolutional layers (once the weights convex combination are summed). Note that the Ensemble MC-SGD with computing cost corresponding to 1 on the x-axis corresponds to the bagging strategy where incoming batches are randomly assigned to 1 one the member of the ensemble (each of them eventually gets to see $1/n$ of the total dataset).

Overall, depending on the limiting factor and use case, one can either use Vanilla Ensemble for its simplicity and performance regardless of the compute cost. However, Subspace-Connectivity has the best cost-performance trade-off among other methods.

6 Related Work

The methodology to tackle catastrophic forgetting for continual learning has been extensively populated with three main groups.

Continual learning. Among pioneer methods to alleviate the catastrophic forgetting, we can name *regularization-based* methods that limit the drift in important parameters of features of past tasks (Kirkpatrick et al., 2017; Zenke et al., 2017; Nguyen et al., 2017; Yin et al., 2020). For instance, EWC (Kirkpatrick et al., 2017) uses the Fisher information to identify the important parameters. A major drawback of the regularization methods is that they often need multiple passes over data to perform well (Chaudhry et al., 2019a) and when the number of tasks is large, they suffer more from the feature drift (Titsias et al., 2020).

The second group of methods in continual learning is the *memory-based* methods that keep a small episodic memory of data from past tasks to either replay those examples (Chaudhry et al., 2019b; Rebuffi et al., 2017; Riemer et al., 2018; Lesort, 2020) or use them for improving the optimization procedure such as projection methods (Farajtabar et al., 2020; Bennani et al., 2020; Saha et al., 2021), or train a generative model to serve that purpose (Shin et al., 2017; Kirichenko et al., 2021). A-GEM (Chaudhry et al., 2019a) is a notable example of these methods that use gradients of past tasks to modify the gradients of the new task and alleviate the forgetting.

Finally, *parameter isolation* methods focus on the neural network modules that can be either be expanded for each new task (Aljundi et al., 2017) or a sub-network will be allocated for each task (Wortsman et al., 2020; Fernando et al., 2017), or create implicit gateways for different tasks (Mirzadeh et al., 2020a). However, the expansion-based methods' memory and compute requirement grows as the number of tasks grows. In addition, these methods rely on the task identifiers for selecting the appropriate module for prediction and often cannot operate without this information.

Perhaps our work is mainly related to regularization- and memory-based methods. Our proposed algorithm maintains a memory of past data for regularization purposes (i.e., encouraging the connectivity between subspaces across tasks).

Ensemble methods for Continual Learning. Ensemble models have been utilized in the literature on continual learning to mitigate catastrophic forgetting, as demonstrated in studies such as Cano & Krawczyk (2022); Wang et al. (2022); Li et al. (2020). For instance, Li et al. (2020) utilized distillation to maintain knowledge from previous experiences by ensembling past predictions with the current model. In Cano & Krawczyk (2022), a module was designed to detect data drifts, and an ensemble cooperation was used to distribute the data knowledge among each member. Wang et al. (2022) demonstrated that an ensemble of small models can outperform a larger one, but they mainly concentrated on parameter efficiency, which still incurs a linear compute cost, whereas our approach focuses on decreasing the compute cost. Although these studies also utilized ensemble methods, the direct comparison of compute cost is not always straightforward, and we aim to provide a comprehensive perspective on ensemble models dynamic in continual learning rather than ensembling specific algorithm.

Mode connectivity. Draxler et al. (2018); Garipov et al. (2018) studied the loss landscape and existence of connectivity of neural network solutions. They discovered the existence of pathways/curves of non-increasing loss between solutions optima. In the context of continual learning, Mirzadeh et al. (2021) have recently shown that multitask and continual minima are connected via low-loss paths and leveraged this property to connect tasks' minima in continual learning and proposed the MC-SGD algorithm, which encourages the linear connectivity between tasks' minima via path regularization. While our work is inspired by their findings, we note that MC-SGD is developed with a single model continual learning in mind, while we extend their work for the continual learning setup with multiple models.

Neural network subspaces. Wortsman et al. (2021); Benton et al. (2021) recently proposed to connect solution of an ensemble of models through a region in the weight space known as *subspace*. That region is a low loss surface and shown to outperform the solution of ensemble models. Gaya et al. (2021) learned a subspace of policies in the reinforcement learning context for fast adaptation. While our work is directly motivated by the subspace literature, we note that these methods have been studied in single-task settings. In contrast, our work extends them for continual learning with a sequence of tasks rather than a single task.

7 Conclusion

While continual learning literature focuses mainly on studying the problem with a single model, we have extended it to the multiple models' case in this work. To the best of our knowledge, we are the first to investigate the behavior of ensemble methods in continual learning. Due to their enhanced diversity power, vanilla ensemble achieves good performance, but this comes at the expense of high computational cost growth. To overcome this challenge, and inspired by the recent advances in the mode connectivity literature, we have proposed a simple yet computationally efficient algorithm to improve the performance of subspace ensemble method. We believe it is important not to consider this seemingly simple method as the main contribution of our work but rather the understanding and implication of ensemble methods for continual learning.

Despite the simplicity, ensemble methods provide a reliable mechanism to mitigate catastrophic forgetting in continual learning. The choice of an ensemble method for continual learning highly depends on the use case and limiting factors (e.g., performance and cost trade-off). We believe our work can be a stepping-stone for several future works, such as further studying the interactions between multiple models in continual learning and designing more efficient ensembling techniques.

Limitation. One limitation of our work is the dependence on a (small) sample of previous tasks to ensure the connectivity. Another limitation may arise due to the nature of mode connectivity and subspace methods because they decrease the capacity of the model for learning new tasks as an expense for increased stability. Finding the right trade-off between forgetting improvement (aka stability of previous knowledge) and learning accuracy (plasticity for learning new knowledge) is very interesting line of work for future. In this regard, one may look for explicitly employing parameter isolation or dynamic expansion methods (like adding new member to the ensemble) to cover for the decreased capacity of learning combined with a non-forgetting continual learning algorithm.

REFERENCES

Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7120–7129, 2017.

Mehdi Abbana Bennani, Thang Doan, and Masashi Sugiyama. Generalisation guarantees for continual learning with orthogonal gradient descent. *arXiv preprint arXiv:2006.11942*, 2020.

Gregory W. Benton, Wesley Maddox, Sanae Lotfi, and Andrew Gordon Wilson. Loss surface simplexes for mode connecting volumes and fast ensembling. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 769–779. PMLR, 2021.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Lucas Caccia, Jing Xu, Myle Ott, Marcaurelio Ranzato, and Ludovic Denoyer. On anytime learning at macroscale. In Sarath Chandar, Razvan Pascanu, and Doina Precup (eds.), *Proceedings of The 1st Conference on Lifelong Learning Agents*, volume 199 of *Proceedings of Machine Learning Research*, pp. 165–182. PMLR, 22–24 Aug 2022. URL https://proceedings.mlr.press/v199/caccia22a.html.

Alberto Cano and Bartosz Krawczyk. Rose: Robust online self-adjusting ensemble for continual learning on imbalanced drifting data streams. *Machine Learning*, 04 2022. doi: 10.1007/s10994-022-06168-x.

Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 18, 2004.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-GEM. In *International Conference on Learning Representations*, 2019a.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019b.

Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pp. 1–15. Springer, 2000.

Thang Doan, Mehdi Abbana Bennani, Bogdan Mazoure, Guillaume Rabusseau, and Pierre Alquier. A theoretical analysis of catastrophic forgetting through the ntk overlap matrix. In *International Conference on Artificial Intelligence and Statistics*, pp. 1072–1080. PMLR, 2021.

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *International Conference on Machine Learning*, pp. 1308–1317, 2018.

Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 3762–3773. PMLR, 2020.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2019.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Jean-Baptiste Gaya, Laure Soulier, and Ludovic Denoyer. Learning a subspace of policies for online adaptation in reinforcement learning. *arXiv preprint arXiv:2110.05169*, 2021.

Ian J. Goodfellow, Mehdi Mirza, Xia Da, Aaron C. Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgeting in gradient-based neural networks. *CoRR*, abs/1312.6211, 2014a.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014b.

Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew M Dai, and Dustin Tran. Training independent subnetworks for robust prediction. *arXiv preprint arXiv:2010.06610*, 2020.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

Jared Kaplan. Notes on contemporary machine learning for physicists. In *"*, 2019.

Polina Kirichenko, Mehrdad Farajtabar, Dushyant Rao, Balaji Lakshminarayanan, Nir Levine, Ang Li, Huiyi Hu, Andrew Gordon Wilson, and Razvan Pascanu. Task-agnostic continual learning with hybrid probabilistic models. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, and et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, Mar 2017. ISSN 1091-6490. doi: 10.1073/pnas.1611835114. URL http://dx.doi.org/10.1073/pnas.1611835114.

Alex Krizhevsky et al. Learning multiple layers of features from tiny images. *CoRR*, 2009.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in Neural Information Processing Systems*, 30, 2017.

Timothée Lesort. Continual learning: Tackling catastrophic forgetting in deep neural networks with replay processes. *arXiv preprint arXiv:2007.00487*, 2020.

Timothée Lesort, Thomas George, and Irina Rish. Continual learning in deep networks: an analysis of the last layer. *arXiv preprint arXiv:2106.01834*, 2021.

Zhuoyun Li, Changhong Zhong, Ruixuan Wang, and Wei-Shi Zheng. Continual learning of new diseases with dual distillation and ensemble strategy. In Anne L. Martel, Purang Abolmaesumi, Danail Stoyanov, Diana Mateus, Maria A. Zuluaga, S. Kevin Zhou, Daniel Racoceanu, and Leo Joskowicz (eds.), *Medical Image Computing and Computer Assisted Intervention – MICCAI 2020*, pp. 169–178, Cham, 2020. Springer International Publishing. ISBN 978-3-030-59710-8.

Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

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

Seyed Iman Mirzadeh, Mehrdad Farajtabar, and Hassan Ghasemzadeh. Dropout as an implicit gating mechanism for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 232–233, 2020a.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. Understanding the role of training regimes in continual learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 7308–7320, 2020b.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. Linear mode connectivity in multitask and continual learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Fmg_fQYUejf.

Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning. In *International Conference on Learning Representations*, 2017.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2017, pp. 5533–5542, 2017.

Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International Conference on Learning Representations*, 2018.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=3AOj0RCNC2.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pp. 2990–2999, 2017.

Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. *arXiv preprint arXiv:2106.14405*, 2021.

S. Thrun. A lifelong learning perspective for mobile robot control. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, volume 1, pp. 23–30 vol.1, 1994. doi: 10.1109/IROS.1994.407413.

Michalis K. Titsias, Jonathan Schwarz, Alexander G. de G. Matthews, Razvan Pascanu, and Yee Whye Teh. Functional regularisation for continual learning with gaussian processes. In *ICLR 2020 : Eighth International Conference on Learning Representations*, 2020.

Liyuan Wang, Xingxing Zhang, Qian Li, Jun Zhu, and Yi Zhong. Coscl: Cooperation of small continual learners is stronger than a big one. In Shai Avidan, Gabriel J. Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner (eds.), *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XXVI*, volume 13686 of *Lecture Notes in Computer Science*, pp. 254–271. Springer, 2022. doi: 10.1007/978-3-031-19809-0_15. URL https://doi.org/10.1007/978-3-031-19809-0_15.

Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Sklf1yrYDr.

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. In *Advances in Neural Information Processing Systems*, volume 33, pp. 15173–15184, 2020.

Mitchell Wortsman, Maxwell Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning neural network subspaces. In *International Conference on Machine Learning*, 2021.

Dong Yin, Mehrdad Farajtabar, Ang Li, Nir Levine, and Alex Mott. Optimization and generalization of regularization-based continual learning: a loss approximation viewpoint. *arXiv preprint arXiv:2006.10974*, 2020.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence, 2017.

Appendix

The appendix is organized as follows:

- Appendix A provides details on each of the Ensemble method (Vanilla, Subspace and Batch Ensemble)
- Appendix C compares the computational cost implications of Vanilla and Subspace Ensemble method.
- Appendix B provides pseudo-code of algorithms used in our ablations analysis and final experiments section.
- Appendix D shows setting of our additional ablation analysis and provides additional insights on the diversity enhanced of Vanilla Ensemble method (Section 3).
- Appendix D.5 provides an in-depth investigation of subspace learning.
- Appendix E.2 provides detailed ablation results and comparison among Ensemble and Scaled models.
- Appendix F details the experimental setup of our final results on 20 tasks for MNIST and CIFAR-100 dataset (Section E)

A Ensemble Methods for Continual Learning

A.1 Training Vanilla Ensemble Models for Continual Learning

In this work, we adapt the learning of ensemble methods to continual learning scenarios. Unless mentioned, we use the standard training procedure, which consists of training independently each of the n models on the same dataset[3]. Given a set of weights $\{\omega_i\}_{i=1}^{n}$, a task τ, when a data batch (x, y) arrives, we optimize $\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_\tau(f_{\omega_i}(x), y)$. Therefore, the models' training differs only in the weights initialization of each model. The average prediction of each model is used as the output of the ensemble for evaluation ($\frac{1}{n}\sum_{i=1}^{n} f_{\omega_i}(.)$)[4]. That being said, the training cost simply grows linearly with the number of models n. A pseudo-code of the Ensemble CL Algorithm is provided in Alg. 3 in the Appendix.

A.2 Learning a Subspace Solution for Continual Learning

For the subspace method training, we proceed similarly as in Wortsman et al. (2021). Given a predictor f, a set of n learnable parameters $\{\omega_i\}_{i=1}^{n}$, learning a subspace of dimension n consists in training the predictor $f_{\bar{\omega}}$ as accurate as possible (with $\bar{\omega} = \sum_{i=1}^{n}\alpha_i\omega_i$, $\boldsymbol{\alpha} \in \Delta^n$). Simply put, given a task τ, when a data batch (x, y) arrives, we optimize $\mathcal{L}_\tau(f_{\bar{\omega}}(x), y)$ where $\bar{\omega} = \sum_{i=1}^{n}\alpha_i\omega_i$, with $\boldsymbol{\alpha} \sim \mathcal{U}(\Delta^n)$. Although, one can learn a distribution over the $\boldsymbol{\alpha}$'s, we consider the standard case like in Wortsman et al. (2020) where we sample it uniformly in the simplex Δ^n. The prediction steps differs only in the choice of $\boldsymbol{\alpha}$ and will be discussed in the following sections. A pseudo-code of the Subspace Continual Learning Algorithm is provided in Alg. 4 in the Appendix.

When it comes to backpropagation, subspace methods enjoy slightly similar computation cost as the single model. Given a loss $\mathcal{L}$, the update w.r.t. each ω_i, is (assuming standard SGD optimizer): $\frac{\partial \mathcal{L}}{\partial \omega_i} = \frac{\partial \mathcal{L}}{\partial \bar{\omega}}\frac{\partial \bar{\omega}}{\partial \omega_i} = \alpha_i\frac{\partial \mathcal{L}}{\partial \bar{\omega}}, \forall i = 1...n$. Then, only a *unique* backpropagation through the model f (w.r.t. to $\bar{\omega}$) is needed. A detailed discussion about the computational cost implication between ensemble and subspace methods is provided in Appendix C.

A.3 Batch Ensemble Solution for Continual Learning

Let's denote a weight $\omega \in \mathbb{R}^{m \times p}$ where m and p are respectively the input and output dimension. Wen et al. (2020) define a (common) slow weights ω and a set of fast weights $\{s_i, r_i\}_{i=1}^{n}$ (with $s_i \in \mathbb{R}^m, r_i \in \mathbb{R}^p, \forall i = 1...n$. Each member of the ensemble $i = 1...n$ owns its set of weights $\omega_i = \omega \circ f_i$ with $f_i = r_i s_i^T$. Whenever an incoming batch arrive (x, y), each element of the batch is randomly assigned to a member of the ensemble such that the forward pass (for a given layer) can be vectorized as: $\phi((x \circ R)\omega \circ S)$ where rows of R (respectively S) consists of the vector r_i (respectively s_i) for all examples of the batch x and ϕ the non-linear activation layer. We refer reader to Section 3.1 of Wen et al. (2020) for more details.

[3] We also compare against bagging strategy in Appendix E.2.

[4] Classical majority voting strategies has been tried without significant difference in performance

B Algorithms pseudo-code

Algorithm 2 Vanilla Continual Learning (Single Model)

```
Input  : A task sequence $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_T$, loss function $\mathcal{L}$
Initialize weights $\omega_0^*$
for tasks $\tau = 1, 2, 3, \ldots T$ do
    for $(x, y) \in \mathcal{T}_\tau$ do
        Get $f_{\omega_{\tau-1}}(x)$
        Optimize $\mathcal{L}(f_{\omega_{\tau-1}}(x), y)$  // Fine-tune on task $\tau$
    end for
    Return $\omega_\tau^*$
end for
Output : $\omega_T^*$
```

Algorithm 3 Vanilla Ensemble Continual Learning

```
Input  : A task sequence $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_T$, loss function $\mathcal{L}$, number of models $n$
Initialize set of weights $\{\omega_{i,0}\}_{i=1}^n$
for tasks $\tau = 1, 2, 3, \ldots T$ do
    for $(x, y) \in \mathcal{T}_\tau$ do
        Get $f_{\omega_{i,\tau-1}}(x), \forall i = 1...n$
        Optimize $\frac{1}{n}\sum_{i=1}^n \mathcal{L}(f_{\omega_{i,\tau-1}}(x), y)$  // Update each model independently: requires $n$ backward passes
    end for
    Return $\{\omega_{i,\tau}^*\}_{i=1}^n$
end for
Output : $\{\omega_{i,T}^*\}_{i=1}^n$
```

Algorithm 4 Subspace Ensemble Continual Learning

```
Input  : A task sequence $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_T$, loss function $\mathcal{L}$, number of models $n$
Initialize set of weights $\{\omega_{i,0}\}_{i=1}^n$
for tasks $\tau = 1, 2, 3, \ldots T$ do
    for $(x, y) \in \mathcal{T}_\tau$ do
        $\boldsymbol{\alpha} \sim \mathcal{U}(\Delta^n), \bar{\omega}_{\tau-1} = \sum_i \alpha_i \omega_{i,\tau-1}$  // Sample uniformly convex combination of weights
        Get $f_{\bar{\omega}_{\tau-1}}(x)$
        Optimize $\mathcal{L}(f_{\bar{\omega}_{\tau-1}}(x), y)$
    end for
    Return $\{\omega_{i,\tau}^*\}_{i=1}^n$
end for
Output : $\{\omega_{i,T}^*\}_{i=1}^n$
```

Algorithm 5 Batch Ensemble Continual Learning

```
Input   : A task sequence $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_T$, loss function $\mathcal{L}$, number of models $n$
Initialize set of weights $\{\omega_{i,0}\}_{i=1}^{n}$
 for tasks $\tau = 1, 2, 3, \dots T$ do
    for $(x, y) \in \mathcal{T}_\tau$ do
        for $eachlayer$ do
            Get $R$ and $S$ using $r_i$ and $s_i$, $i = 1 ... n$  // Assign each element of the batch randomly to a member of the ensemble
            Compute $W = ((x_{out} \circ R)\omega \circ S)$  // $x_{out}$ being the output of the previous layer
        end for
        Optimize $\mathcal{L}(f_W(x), y)$
    end for
    Return $\{\omega^*_\tau, s_\tau, r_\tau\}_{i=1}^{n}$
end for
Output : $\{\omega^*_{i,T}\}_{i=1}^{n}$
```

Algorithm 6 Subspace + Experience Replay (ER)

```
Input   : A task sequence $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_T$, loss function $\mathcal{L}$, number of models $n$, replay buffer $\mathcal{B}$, $m_{\mathcal{B}}$ memory size per task
Initialize set of weights $\{\omega_{i,0}\}_{i=1}^{n}$
 for tasks $\tau = 1, 2, 3, \dots T$ do
    for $(x, y) \in \mathcal{T}_\tau$ do
        $\boldsymbol{\alpha} \sim \mathcal{U}(\Delta^n)$, $\bar{\omega}_{\tau-1} = \sum_i \alpha_i \omega_{i,\tau-1}$  // Sample uniformly convex combination of weights
        $(x', y') \sim \mathcal{B}$  // Samples elements from the buffer
        $x \leftarrow$ Concat$(x, x')$
        $y \leftarrow$ Concat$(y, y')$
        Get $f_{\bar{\omega}_{\tau-1}}(x)$
        Optimize $\mathcal{L}(f_{\bar{\omega}_{\tau-1}}(x), y)$
    end for
    Return $\{\omega^*_{i,\tau}\}_{i=1}^{n}$
     for $(x, y) \in \mathcal{T}_\tau$ do
        $\mathcal{B} \leftarrow \mathcal{B} \bigcup \{x, y\}$  // Store $m_{\mathcal{B}}$ elements per class per task in the buffer
    end for
end for
Output : $\{\omega^*_{i,T}\}_{i=1}^{n}$
```

Algorithm 7 Subspace-Connectivity CL

```
Input :A task sequence $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_T$, number of models $n$ , buffer $\mathcal{B}$ and memory size $m_{\mathcal{B}}$
    1. Initialize set of weigths $S^n = \{\omega_{0,i}\}_{i=1}^n$, buffer $\mathcal{B} \leftarrow \{\}$
    2. for tasks $\tau = 1, 2, 3, \ldots T$ do
        Get $\{\hat{\omega}_{\tau,i}^*\}_{i=1}^n$ with Eq 1
          // Learn subspaces solution for task $\tau$
        for $(x, y) \in \mathcal{T}_\tau$ do
            $\mathcal{B} \leftarrow \mathcal{B} \bigcup \{x, y\}$
        end for
        Get $\{\omega_{\tau,i}^*\}_{i=1}^n$ with Eq 2 using samples from $\mathcal{B}$
          // Connect previous solution’ subspaces
    end for
Output: $\{\omega_{T,i}^*\}_{i=1}^n$
```

C Computational cost implications for ensemble and subspace methods

This section discusses the implication cost of Vanilla and Subspace Ensemble methods.

For the ensemble methods, since each model is trained independently on the same dataset, the cost is n times higher (n times more forward and backward passes).

Compared to the single model, the subspace method contains one additional operation for the inference and backpropagation operation. Before the inference, a convex combination of the weights is sampled: $\bar{\omega} = \sum_{i=1}^{n} \alpha_i \omega_i$, $\alpha \sim \mathcal{U}(\Delta^n)$ (weights mixing). For the backpropagation, only one backward pass is needed ($\frac{\partial \mathcal{L}}{\partial \bar{\omega}}$) before assigning the new weights $\omega_i = \omega_i - l_r \alpha_i \frac{\partial \mathcal{L}}{\partial \bar{\omega}}$, $\forall i = 1...n$, l_r being the learning rate.

Overall, the subspace method has only two supplementary addition operations as an overhead cost compared to the single model which is much cheaper than the $n - 1$ additional forward and backward pass of the ensemble methods. This makes the subspace method more efficient.

Method	forward pass	backward pass
Single model	1 inference pass ($f(x, \omega)$)	1 backpropagation pass ($\frac{\partial \mathcal{L}}{\partial \omega}$)
Ensemble	n inference passes ($f(x, \omega_i)$, $\forall i = 1...n$)	n backpropagations passes ($\frac{\partial \mathcal{L}}{\partial \omega_i}$, $\forall i = 1...n$)
Subspace	$\bar{\omega} = \sum_{i=1}^{n} \alpha_i \omega_i$ 1 inference pass ($f(x, \bar{\omega})$)	1 backpropagation pass ($\frac{\partial \mathcal{L}}{\partial \bar{\omega}}$) $\omega_i \leftarrow \omega_i - l_r \alpha_i \frac{\partial \mathcal{L}}{\partial \bar{\omega}}, \forall i = 1...n$

Table 3: Computational cost comparison between different methods. In red are the additional operations compared to the single model. While ensemble methods have $(n - 1)$ additional backward passes (backward pass), subspace methods only incur addition operations as an overhead cost which are much cheaper.

D Setup of ablation and additional results

D.1 Experimental parameters

For the ablation of Section 3, we train on 5 tasks for Rotated MNIST and Split CIFAR-100. The details of the setup are the following:

Rotated MNIST : The incremental angle is 22.5° for a total of 90°. This made the benchmark more challenging as done in Mirzadeh et al. (2020b) with one training epoch per task.

Split CIFAR-100 : We use the 25 first tasks of CIFAR-100 and train for 5 epochs per task to highlight the properties of ensemble methods.

The hyperparameters for each baseline are the following:

Naive SGD and Vanilla Ensemble (VE) and Batch Ensemble (BE)

- learning rate: [0.2, 0.15, **0.1** (MNIST), **0.05** (CIFAR-100), 0.01]
- learning rate decay: [1.0 (CIFAR-100)[5], 0.95, 0.9, 0.5 (MNIST)]
- batch size: [10,32,64,128]

Subspace Ensemble (SE)

- standardized learning rate[6]: [0.2, **0.15** (CIFAR-100), **0.1** (MNIST)]
- learning rate decay: [1.0 (CIFAR-100), 0.95, 0.9, 0.5 (MNIST)]
- batch size: 10

An other important aspect of subspace methods are how close each model is initialized. The further away from each other they are, the more iteration will be needed to update the whole volume. To control this volume, we initialize each model's weight with respect to the first one as: $\omega_i = \omega_1 * \mathcal{N}(1, \sigma_{init}), \forall i = 1...n$, ω_1 being the initialized weight of the first model. The variance σ_{init} controls this volume. For MNIST, we use $\sigma_{init} = 1.0$ ($n = 2, 3$) and $\sigma_{init} = 1.5$ ($n = 5, 10$), for CIFAR-100 we use $\sigma_{init} = 0.1$ ($n = 2, 3, 5, 10$).

D.2 Metrics

To quantify the properties of ensemble and subspace methods, we have used the following metrics: **Learning Accuracy** $LA_T = \frac{1}{T}\sum_{\tau=1}^{T} a_{\tau,\tau}$, where $a_{\tau,\tau}$ represents the accuracy on task τ after learning on task τ the first time, **Forgetting** $F_T = \frac{1}{T-1}\sum_{\tau=1}^{T-1} \max_{t=\{1..T-1\}}(a_{t,\tau} - a_{T,\tau})$ and **Forgetting Improvement** $FI_T = F_T(\textit{single model}) - F_T(\textit{ensemble/subspace model})$

[5] A value of 1 means no decay has been applied to the learning rate

[6] This is an average learning rate per model, i.e we use a learning rate of 0.3 when using $n = 3$ models for MNIST

D.3 Diversity of performance of the Vanilla Ensemble methods

To measure the diversity of each member in the ensemble method (Ensemble Continual Learning Alg. 3), we show the boxplot of the final average accuracy for each member of the ensemble (line) on each task (column) in Figure 7. The x-axis represents each individual in the ensemble while the last index represents the final prediction of the ensemble (noted "Avg"). As we increase the number of models, the gap between the highest accuracy of each individual and the final prediction increases (For instance, compare gap between blue and gray boxplot for the 3rd column). The diversity in each model (initialized differently) might explain good performance of ensemble methods.

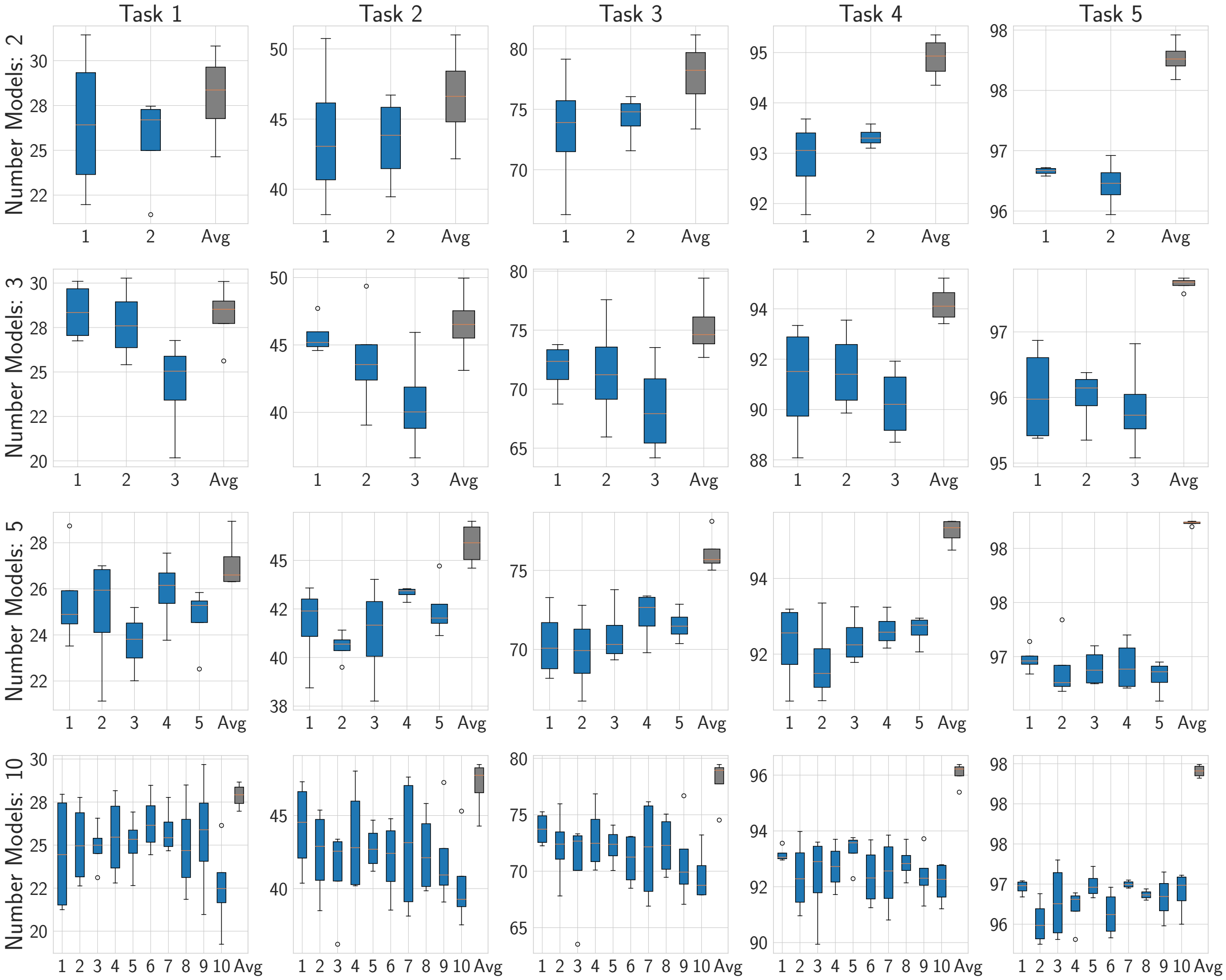


Figure 7: Final accuracy for each model of the ensemble (blue) and the final ensemble prediction (gray) for each task for Rotated MNIST. The final ensemble prediction accuracy is almost always higher than the best accuracy of the best model. As we increase the number of models, we can see a diversity in the prediction of each model, each of them seems to specialize naturally in diverse tasks (look at $n = 5, 10$). This might contribute to the good performance of the ensemble method.

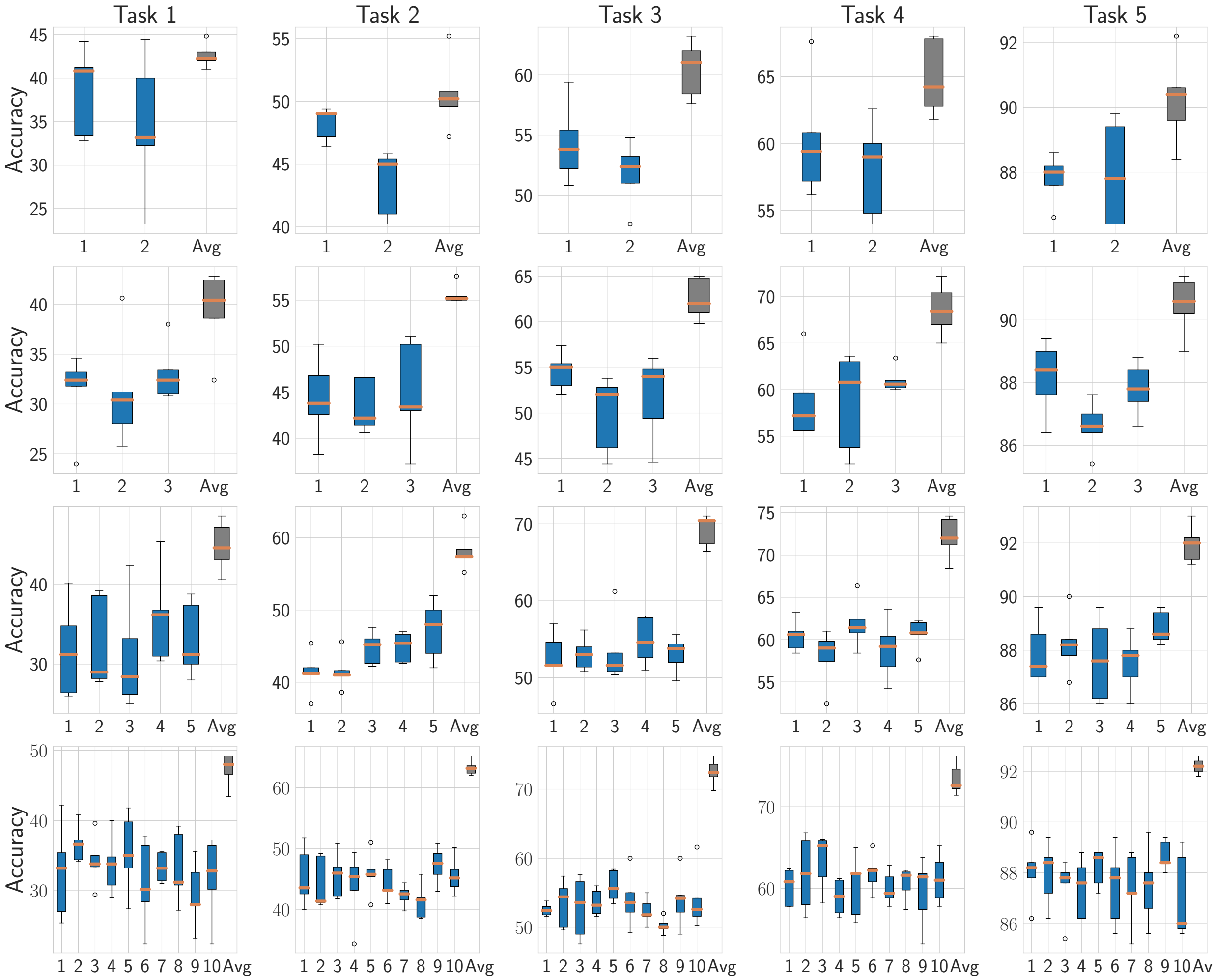


Figure 8: Final accuracy for each model of the ensemble (blue) and the final ensemble prediction (gray) for each task for Split CIFAR-100. Note how the ensemble accuracy (grey) is almost always much beter than the accuracy of the best member (blue).

D.4 Diversity in prediction of Vanilla Ensemble methods

We here provide a visualization of the prediction's evolution throughout the learning experience for the 5 tasks ablation on Split CIFAR-100 for Vanilla Ensemble (average prediction). Figure 9 shows the output of each member of an ensemble for a given sample of task 1 (the column represents the weights ω_i^* corresponding to the current task i). The y-axis represents the possible prediction (class 1 to 5) with the Ground truth being highlighted in orange (class 1 here). The dashed red square is the final prediction made by the ensemble.

Throughout the learning, one can see the forgetting about solution's 1 prediction that leads to diversity in prediction (read the 2nd row for instance). This obviously leads to forgetting of former's solution. The more member an ensemble has (take last row for instance), the more likely it will be able to keep its former's solution has a main core member sticks around the former's solution (compare 4th column).

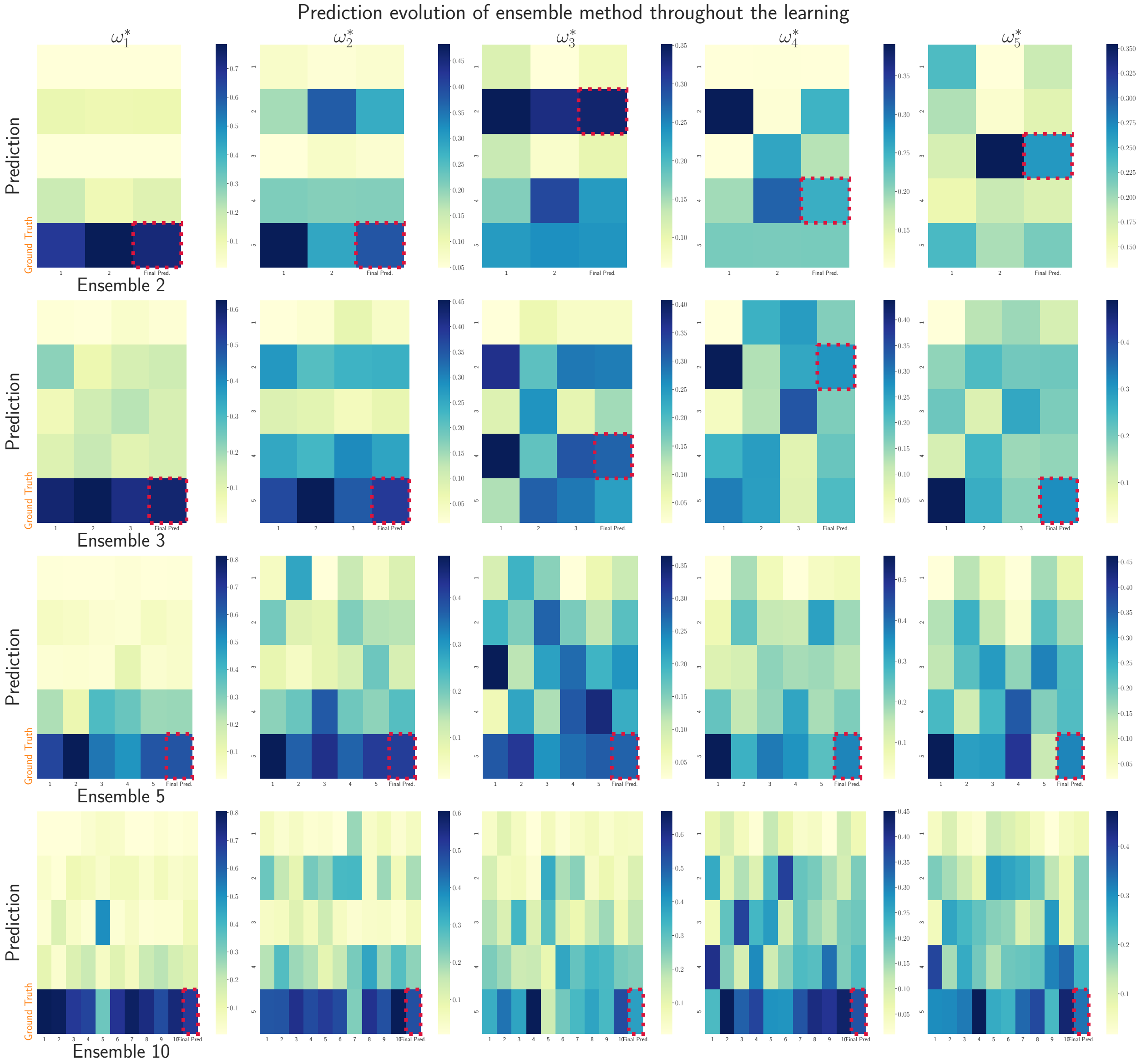


Figure 9: Evolution of prediction for Split CIFAR-100 on one sample of task 1. The y-axis represents the prediction for each class 1 to 5 while the x-axis represents the behavior of member i. Throughout the training (read from left to right), predictions of former's solution is forgotten leading to diversity as well. The more member an ensemble has, the more likely the former solution's can be remembered since the average prediction is used as the final output (look at last row 4th and 5th column).

D.5 Understanding subspace properties

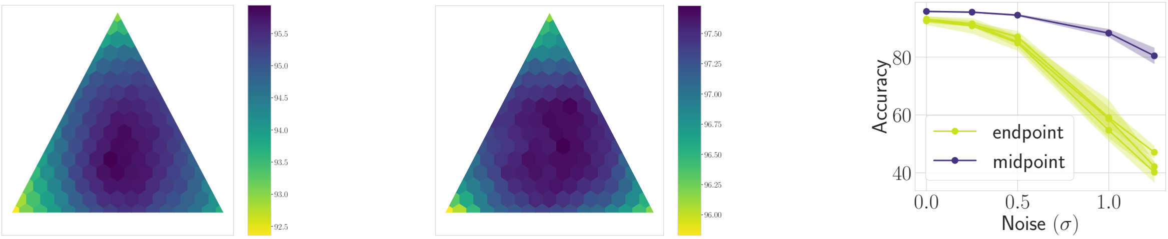
This section aims at showing the dynamic of subspace through the training process. We will show that applying naively Subspace (SE) and adding Experience Replay (ER) still show high forgetting.

To this end, we raise two questions: (1) "What are the interesting properties of the learned subspaces in continual learning?" and (2) "How do subspaces evolve throughout the learning experience?". In Sec. D.6, we study the first question and highlight an important property of the subspace method, that is, the center of the subspace contains more accurate and robust solutions. To answer the second question, in Sec. D.7, we show the limit of the subspace method that can still suffer from the forgetting problem when the number of tasks increases.

For this purpose we design Subspace-Connectivity exploits the connectivity of the subspaces throughout the continual learning. By connectivity, we mean that there is a path between two solutions along which the loss value and the test error stay low Draxler et al. (2018); Garipov et al. (2018).

D.6 Subspace midpoint gets the best accuracy

In this section, we further investigate the behavior of the learned subspaced ensembles by monitoring the accuracy within the subspace.



(a) Learning accuracy of task 1

(b) Learning accuracy of task 10

(c) Robustness to Gaussian noise around solution's minima.

Figure 10: Rotated MNIST: Learning Accuracy for different tasks inside a subspace region with 3 models. Midpoints (center) show the best performance as opposed to the endpoints (corners) **(left, middle)**. Robustness to uniform noise for the case of $n = 3$ models. Midpoint is more robust to weight perturbation than endpoints **(right)**

To this end, we use Subspace Continual Learning algorithm (Alg. 4) with $n = 3$ models. In Figs. 10a and 10b we show the accuracy across the subspace at the beginning of learning (task 1) and in the middle of learning (task 10), respectively. The plots illustrate that the center (midpoint) has higher learning accuracy within each region. This also translates into more robustness of the midpoint versus the endpoints (Fig. 10c).

D.7 Subspaces still forget

Now, we shift our focus on the evolution of the learned subspaces throughout the continual learning experience. To this end, we compare the subspace method (Alg. 4) with and without Experience Replay, ER Riemer et al. (2018), (Alg. 6) . Fig. 11 shows snapshots of task 1's accuracy across different times of the training (task $\tau = 1, 5, 10$) on Rotated MNIST (20 tasks). We can observe that while adding ER improves the accuracy, over long sequences of tasks, there is still a degradation of performance. For instance, on task 1 Subspace + ER incurs a performance decrease to 55% (last column ω^*_{10}).

To investigate this performance degradation, we employ the recent technique introduced by Mirzadeh et al. (2021) where the authors show that the linear mode connectivity can explain the performance gap between continual and multitask solutions. Since the subspace method is inherently motivated by the mode connectivity, we believe investigating the connectivity across subspaces can explain the performance degradation. Hence, we visualize the mode connectivity between successive solutions (represented by midpoints) of task 1 and task 2, evaluated on the loss of task 1. In Fig. 12 we can see that the midpoints of subsequent subspaces' solutions are not *linearly connected* meaning the linear interpolated weights between these two solutions does not stay in a low-loss region Mirzadeh et al. (2021). Fig 13 in Appendix provides more detailed analysis and metrics (including accuracy) with the same conclusion.

Thus far, we have investigated the properties of the subspaces and their evolution throughout the learning experience.

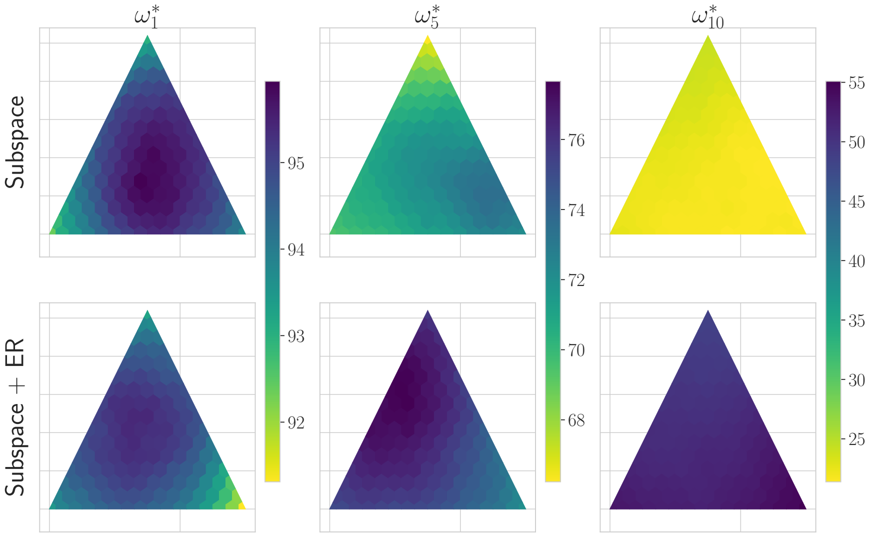


Figure 11: Rotated MNIST: The evolution of accuracy of task 1 throughout the training. Even though the subspace method restricts weight's movement, there is still a noticeable degradation in performance (hence a large amount of forgetting).

By tracking the evolution of the subspace, we have also observed a performance degradation, which can be explained by the lack of connectivity between solutions. These results motivate us to design Subspace-Connectivity that takes these findings into action. In the next section, we highlight the improve in mode connectivity of our algorithm against Subspace Ensemble and Subspace + ER.

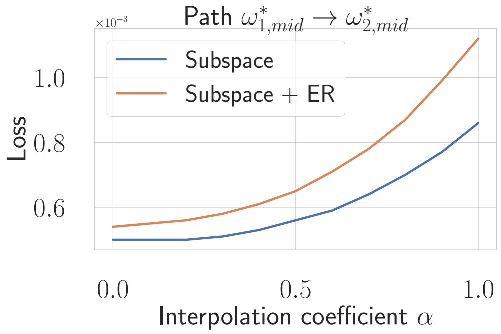


Figure 12: Rotated MNIST: The sequential subspace solutions are not linearly connected.

D.7.1 Subspace connectivity

To measure the mode connectivity Mirzadeh et al. (2021) between subsequent solutions, we evaluate the loss function between interpolated solutions $\alpha\omega_i^* + (1-\alpha)\omega_{i+1}^*, \forall\alpha \in [0,1]$. Fig 13 provides a comparison between our method Subspace-Connectivity against Subspace CL (Alg. 4) and Subspace + ER (Alg. 6). We clearly see that the two latter incurs a high variation in their loss and accuracy between successive solutions on task 1'loss. As an example, while Subspace-Connectivity incurs a slight variation of 1% in accuracy when interpolating between ω_2^* and ω_3^* (2nd column) the other methods incur a drop of more than 2% of accuracy. This drop is even larger when interpolating between ω_3^* and ω_4^* (3rd column). Now we are interested to investigate the mode connectivity for further task's solution of Subspace-Connectivity .

Figure 14 shows the linear connectivity between solutions more than 5 tasks further in time. Although performance might have decreased between ω_{11}^* and ω_2^* on task 1, we can see that the linear path between ω_{11}^* and ω_{12}^* (red line) is close to horizontal losing roughly 2% of accuracy (first column) and 1% if one considers the path between ω_6^* and ω_7^* (green line). Note the high accuracy of $\omega_6^* \to \omega_7^*$ on task 1 (first column, green line) which reaches $\sim 94\%$ of accuracy while the accuracy of the first task (with ω_1^*) was around 96%.

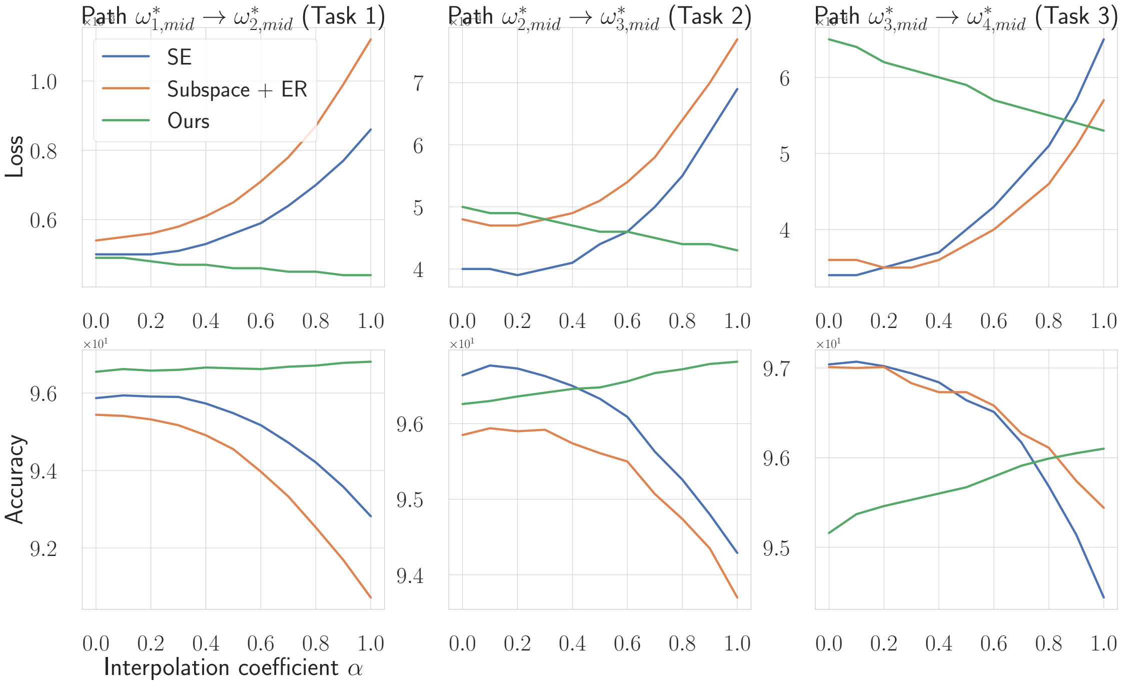


Figure 13: Connectivity comparison between Subspace-Connectivity and other baselines. Naive and ER+Subspace incurs high variation when interpolating between successive solutions since they do not enforce mode connectivity.

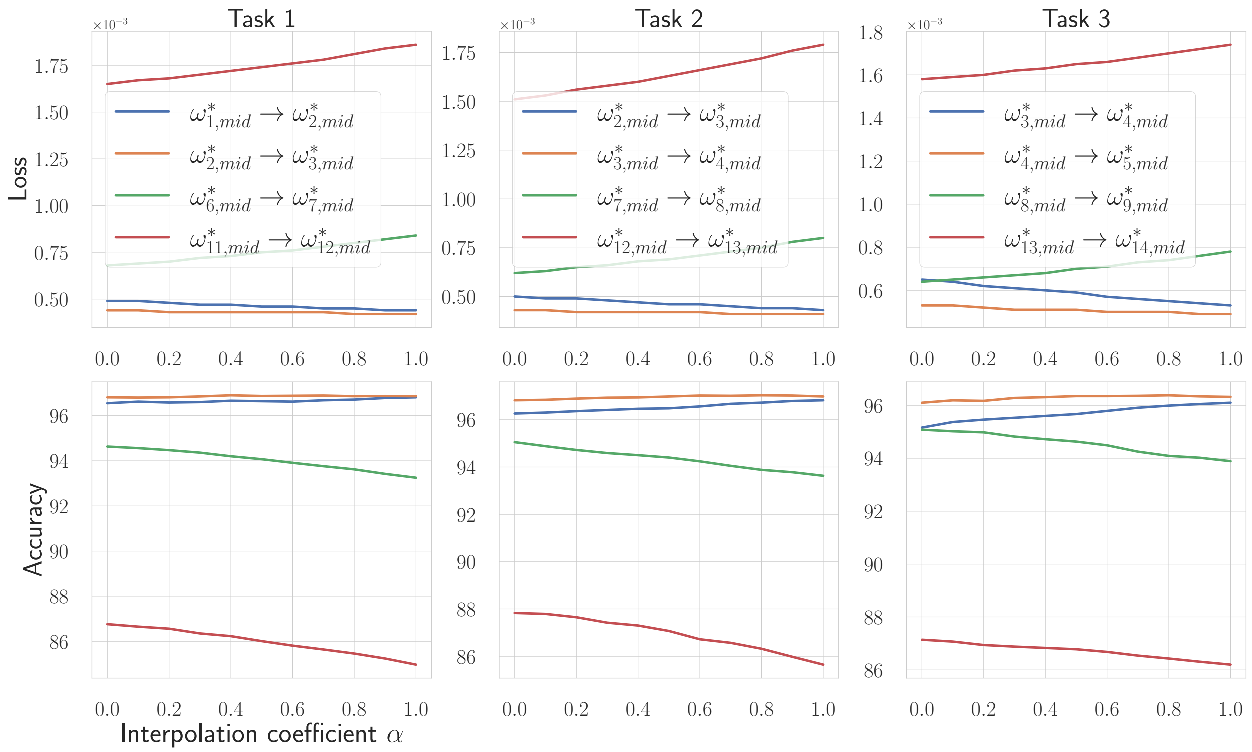
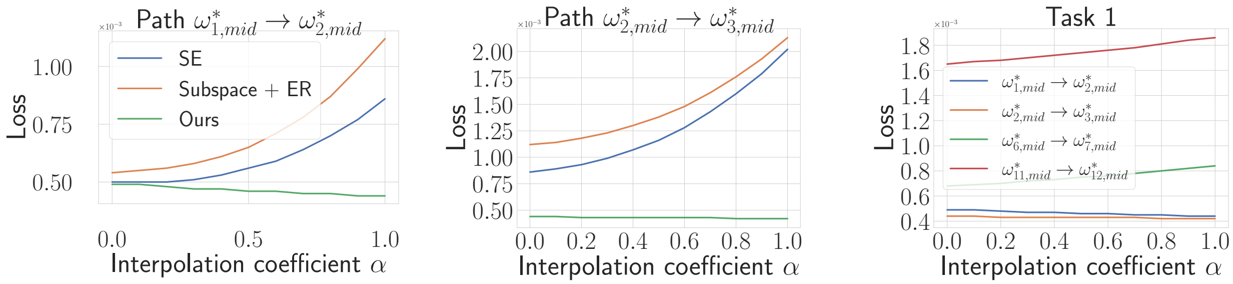


Figure 14: Subsequent solutions of Subspace-Connectivity maintain more or less connectivity (horizontality of each curve).



(a)

(b)

(c)

Figure 15: While applying naively subspace (Subspace and Subspace + ER) does not allow mode connectivity (loss function is increasing between two subsequent solutions) (left), our method (denoted "Ours") allows linear mode connectivty between successive solution'.

E Additional results and setup details

In this section, we first provide our experimental setup and benchmarks used. We then check if our method connects the subspace of previous solutions together and finally compare our algorithm Subspace-Connectivity against the main baselines of the literature.

E.1 Experimental setup

Setup The experimental setup, such as benchmarks, network architectures, continual learning setting (e.g., number of tasks, episodic memory size, and training epochs per task), hyper-parameters, and evaluation metrics are chosen to be similar to several other studies Chaudhry et al. (2019a); Mirzadeh et al. (2020b); Chaudhry et al. (2019b); Farajtabar et al. (2020). For all experiments, we report the average and standard deviation over five runs with different random seeds.

Benchmarks We use the standard benchmarks similarly to Goodfellow et al. (2014a) and Chaudhry et al. (2019b). Permuted-MNIST Goodfellow et al. (2014a) consists of a series of MNIST supervised learning tasks, where the pixels of each task are permuted with respect to a fixed permutation. Rotation-MNIST Farajtabar et al. (2020) consists of a series of MNIST classification tasks, where the images are rotated with respect to a fixed angle, monotonically. For both MNIST dataset the task ID does not need to be provided since it is a domain incremental task. We increment the rotation angle by 9 degrees at each new task. Split CIFAR-100 Chaudhry et al. (2019b) is constructed by splitting the original CIFAR-100 dataset Krizhevsky et al. (2009) into 20 disjoint subsets. In order to assess the robustness to catastrophic forgetting over long tasks sequences, all datasets have 20 tasks. Split miniImageNet is a variant of the ImageNet dataset Russakovsky et al. (2015), also splitted in 20 disjoint subsets where each subset is formed by sampling without replacement of 5 classes out of 100. Both CIFAR-100 and miniImageNet contains 20 tasks with 500 samples for each of the 5 classes and request the task ID to be provided as input to the model.

Architectures For MNIST dataset, we used a fully connected neural networks with two hidden layers of 256 ReLU hidden units as in Chaudhry et al. (2019a); Mirzadeh et al. (2020b); Chaudhry et al. (2019b); Farajtabar et al. (2020). For Split CIFAR-100, we used the same reduced Resnet18 as in Mirzadeh et al. (2020b) (with three times less features maps accros all layers). For Split miniImageNet, we adapted the network used with CIFAR-100 by adapting the input dimension of the last fully connected layers since both dataset have different input dimensions ((3,84,84) for miniImageNet versus (3,32,32) for CIFAR-100).

Evaluation metrics To assess the performance of each baseline, we report two metrics used in the literature which are the **Final Accuracy** and **Forgetting Measure**. The Final Accuracy after T tasks is the average validation accuracy over all the tasks $\tau = 1...T$ defined as: $A_T = \frac{1}{T}\sum_{\tau=1}^{T} a_{T,\tau}$ where $a_{T,\tau}$ is the validation accuracy of task τ after the model finished learning on task T. The Forgetting Measure is defined as: $F_T = \frac{1}{T-1}\sum_{\tau=1}^{T-1} \max_{t=\{1..T-1\}}(a_{t,\tau} - a_{T,\tau})$

E.2 Ensemble model ablations

This section provides various ablation experiments between Scaled MC-SGD, Ensemble MCSGD and Subspace-Connectivity :

- Table 4 shows the performance for Subspace-Connectivity with different number of models n
- Table 5 compare Subspace-Connectivity against Scaled single model and Ensemble MC-SGD varying: number of models n, parameters and for different level of compute cost (FLOPS) and different strategies for Vanilla Ensemble (bagging and not bagging).

Scaled MC-SGD In order to compare to the performance on a single model ($n = 1$), we increase its capacity to match the number of parameters of the ensemble methods. While fully connected networks (MNIST dataset), we increase the number of hidden units ($h = 256, 450, 600$), for the Restnet18 network (Split CIFAR-100), we increase the number of channels ($nf = 20, 29, 35$).

Bagging strategy for Ensemble MC-SGD Every time a data batch arrives, we sample uniformly one model among the other to receive the data and be updated. This allows to compare Ensemble MC-SGD at the same level of compute as MC-SGD since each member of the ensemble get to see $1/n$ of the whole dataset.

Calculation of the FLOPS The Floating Point Operation per Second is a metric to quantify the computational cost for an algorithm. Our reported FLOPS metric represents a forward pass cost given a batch size of 10. To have an idea of the total training cost, one can approximate the backward pass as ~ 2 times the forward pass cost Kaplan (2019). The relative FLOPS is the ratio between our reported Inference (forward pass) FLOPS for a given algorithm and its Single version, i.e the reference algorithm is MC-SGD in our case.

Number of model	**Permuted MNIST**		**Rotated MNIST**		**Split CIFAR-100**		**Split miniImageNet**	
	Accuracy ↑	Forgetting ↓	Accuracy ↑	Forgetting ↓	Accuracy ↑	Forgetting ↓	Accuracy ↑	Forgetting ↓
2	87.1 (±0.19)	0.07 (±0.01)	86.6 (±0.45)	0.07 (±0.01)	60.97 (±1.53)	0.05 (±0.01)	57.10 (±1.53)	0.05 (±0.01)
3	87.8 (±0.53)	0.06 (±0.01)	86.7 (±0.67)	0.07 (±0.01)	61.74 (±0.80)	0.05 (± 0.01)	58.17 (±0.84)	0.03 (±0.01)
5	87.8 (±0.3)	0.07 (±0.01)	86.8 (±0.46)	0.07 (±0.01)	60.85 (±0.73)	0.05 (± 0.01)	58.11 (±1.23)	0.03 (± 0.01)

Table 4: Final accuracy for subspace method with a different number of models n. For $n \geq 2$ models, there is not much difference in performance for MNIST dataset while for Split CIFAR-100, $n = 3$ gives the best performance.

Number of model	**Permuted MNIST**		**Rotated MNIST**		**Relative FLOPS ratio**
	Accuracy ↑	Forgetting ↓	Accuracy ↑	Forgetting ↓	
$\lvert\theta\rvert = 268K$					
Scaled MC-SGD (256)	83.17 (±0.63)	0.10 (±0.01)	81.70 (±0.43)	0.08 (±0.01)	1
$\lvert\theta\rvert = 560K$					
Scaled MC-SGD (450)	86.80 (±0.93)	0.07 (±0.01)	82.67 (±0.20)	0.08 (±0.01)	2.1
Ensemble MC-SGD ($n = 2$, bagging*)	86.23 (±0.52)	0.06 (±0.01)	80.7 (±0.50)	0.09 (±0.01)	1
Ensemble MC-SGD ($n = 2$)	86.5 (±0.33)	0.08 (±0.01)	83.00 (±0.45)	0.08 (±0.01)	2
Subspace-Connectivity ($n = 2$)	87.1 (±0.19)	0.07 (±0.01)	86.6 (±0.45)	0.07 (±0.01)	1.1
$\lvert\theta\rvert = 836K$					
Scaled MC-SGD (600)	88.03 (±0.36)	0.06 (±0.01)	83.67 (±0.40)	0.07 (±0.01)	3.1
Ensemble MC-SGD ($n = 3$, bagging)	86.60 (±0.46)	0.05 (±0.01)	80.00 (±0.24)	0.08 (±0.01)	1
Ensemble MC-SGD ($n = 3$)	88.30 (±0.48)	0.06 (±0.01)	83.63 (±0.39)	0.07 (±0.01)	3
Subspace-Connectivity ($n = 3$)	87.8 (±0.30)	0.07 (±0.01)	86.7 (±0.67)	0.07 (±0.01)	1.2

Number of model	**Split CIFAR-100**		**Split miniImageNet**		**Relative FLOPS ratio**
	Accuracy ↑	Forgetting ↓	Accuracy ↑	Forgetting ↓	
$\lvert\theta\rvert = 500K$					
Scaled MC-SGD ($nf = 20$)	58.22 (±0.91)	0.08 (±0.01)	54.80 (±1.04)	0.05 (±0.01)	1
$\lvert\theta\rvert = 1M$					
Scaled MC-SGD ($nf = 29$)	60.12 (±0.97)	0.07 (±0.01)	55.30 (±0.83)	0.06 (±0.01)	2.08
Ensemble MC-SGD ($n = 2$, bagging)	56.87 (±0.80)	0.06 (±0.01)	54.00 (±0.77)	0.05 (±0.01)	1
Ensemble MC-SGD ($n = 2$)	60.83 (±0.99)	0.09 (±0.01)	57.60 (±0.55)	0.04 (±0.01)	2
Subspace-Connectivity ($n = 2$)	60.97 (±1.53)	0.05 (±0.01)	57.10 (±0.79)	0.05 (±0.01)	≈ 1.00
$\lvert\theta\rvert = 1.5M$					
Scaled MC-SGD ($nf = 35$)	60.50 (±0.84)	0.06 (±0.01)	55.44 (±1.36)	0.05 (±0.01)	3.01
Ensemble MC-SGD ($n = 3$, bagging)	55.48 (±1.20)	0.06 (±0.01)	52.39 (±0.60)	0.03 (±0.01)	1
Ensemble MC-SGD ($n = 3$)	64.12 (±1.16)	0.06 (±0.01)	59.1 (±1.1)	0.04 (±0.01)	3
Subspace-Connectivity ($n = 3$)	61.74 (±0.80)	0.05 (± 0.01)	58.17 (±0.84)	0.03 (± 0.01)	≈ 1.00

Table 5: Performance comparison against Scaled MC-SGD, Ensemble MC-SGD and Subspace-Connectivity . Ensemble MC-SGD gets the best performance but at a high compute cost. However, if we compare with a fair compute cost (see bagging) Ensemble MC-SGD performs worst than Subspace-Connectivity .

F Baselines hyperparameters

We first enumerate the hyperparameters used for the 20 tasks experiments in Table 6 before describing in detail the ablation in this section.

F.1 Hyper-Parameters

For the experiment in Section 5, we have used the following grid for each model. We note that for other algorithms (e.g., A-GEM, and EWC), we ensured that our grid contains the optimal values that the original papers reported. If applicable, all baselines used a buffer memory of 1 element per class per task (which translates in a total replay buffer memory of 200 for MNIST dataset and 100 for CIFAR-100 and miniImageNet). We used the same single training epoch per task setting as in Chaudhry et al. (2019b); Mirzadeh et al. (2020b;a).

Naive SGD

- learning rate: [0.25, **0.1** (MNIST), **0.03** (miniImageNet), **0.01** (CIFAR-100), 0.001]
- batch size: 10

EWC

- learning rate: [0.25, **0.1** (MNIST, CIFAR-100), 0.01, 0.001]
- batch size: [64, **10**]
- λ (regularization): [100, **10** (MNIST, CIFAR-100), 1]

A-GEM

- learning rate: [0.1, **0.1** (MNIST), **0.01** (CIFAR-100), 0.001]
- batch size: [64, **10**]

ER-Reservoir

- learning rate: [0.25, **0.1** (MNIST, miniImageNet), **0.01** (CIFAR-100), 0.001]
- batch size: [64, **10**]

Batch Ensemble

Although Wen et al. (2020) reported an optimal value for an ensemble size of $n = 4$, we found out in our setting that $n = 2$ provided the best results.

- learning rate: [0.25, **0.1** (CIFAR-100,miniImageNet, Permuted), 0.01 (Rotated), 0.001]
- learning rate decay: [0.95, **0.9** (CIFAR-100, miniImageNet), 0.85,0.8 (MNIST)]
- batch size: [128, 64, 32, **10**]

Stable SGD

- initial learning rate: [0.25, **0.1** (MNIST, CIFAR-100), 0.01, 0.001]
- learning rate decay: [0.95, **0.9**(CIFAR-100), 0.85,0.8 (miniImageNet), **0.6**(MNIST)]
- batch size: [64, **10**]
- dropout: [**0.25** (MNIST), 0.1,**0.0** (CIFAR-100, miniImageNet)]

MODE CONNECTIVITY SGD

To obtain continual minima (i.e., $\hat{\omega}_1^*$ to $\hat{\omega}_{20}^*$), we use the following hyperparameters:

- initial learning rate: [0.25, **0.1** (MNIST, CIFAR-100), 0.01, 0.001]
- momentum: [0.9, 0.85, **0.8** (MNIST), **0.7** (miniImageNet), **0.4** (CIFAR-100)]
- learning rate decay: [**0.95** (Rotated MNIST, CIFAR-100), **0.9** (miniImageNet), 0.85, **0.8** (Permuted MNIST), 0.7]
- batch size: [**64** (MNIST), 32, **10** (CIFAR-100, miniImageNet)]
- dropout: [**0.25** (Permuted MNIST), 0.1, **0.0** (Rotated MNIST, CIFAR-100, miniImageNet,)]

To obtain $\bar{\omega}_1^*$ to $\bar{\omega}_{20}^*$, we use the following grid:

- number of samples: [10, **5**, 3] for both MNIST and CIFAR experiments.
- learning rate: [0.2, 0.1, **0.05** (MNIST), **0.01** (CIFAR-100, miniImageNet), 0.001].

SUBSPACE-CONNECTIVITY

To obtain the subspace solution from the first step $\{\hat{\omega}_i^*\}_{i=1}^n$ from Eq 1, we used the following hyperparameters:

- initial learning rate: [**0.3**[7] (CIFAR-100), 0.2, **0.15** (miniImageNet) , **0.1** (MNIST)]
- momentum: [0.9, 0.85, **0.8** (Rotated MNIST), **0.4** (Permuted MNIST), **0** (CIFAR-100, miniImageNet)]
- learning rate decay: [**0.95** (Rotated MNIST, CIFAR-100, miniImageNet), 0.9, **0.8** (Permuted MNIST)]
- batch size: [32, **10** (MNIST, CIFAR-100, miniImageNet)]
- dropout: [**0.25** (Permuted MNIST), 0.1, **0.0** (Rotated MNIST, CIFAR-100, miniImageNet)]

The subspace connectivity steps leading to $\{\omega_i^*\}_{i=1}^n$ (Eq 2) used the following hyperparameters:

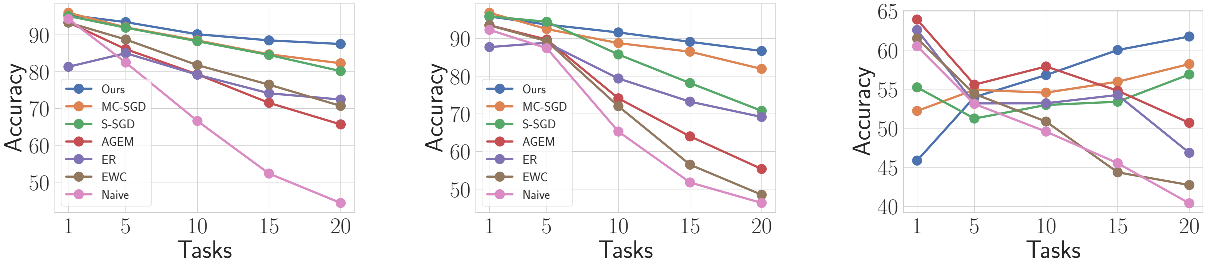
- number of samples: [10, **5** (MNIST), **3** (CIFAR-100, miniImageNet)]
- learning rate: [0.2, **0.1** (CIFAR-100), **0.05** (MNIST, miniImageNet),].

Implementation details of Subspace-Connectivity While the first loss (Eq. 1) is a fine-tuning on the incoming task $\tau - 1$ (Vanilla SGD), the second one (Eq. 2) is done in two steps. First, we initialize the weight using a convex combination of weights around the two former midpoints as $\bar{\omega}_i = \alpha\omega^*_{\tau-1,mid} + (1-\alpha)\hat{\omega}_{\tau,mid}$ then we add multiplicative noise $\bar{\omega}_i * \epsilon, \quad i = 1...n, \epsilon \sim \mathcal{N}(1,\sigma)$ (with $\sigma = 0.005$ for MNIST and $\sigma = 0.01$ for CIFAR-100 and miniImageNet) where $*$ represents element-wise multiplication. The multiplicative noise has the nice property to scale well with the weights magnitude. The α values taken are: [0.9, **0.85** (Rotated MNIT), **0.8** (CIFAR-100), **0.7** (miniImageNet), **0.25** (Permuted MNIST),]

Method	**Permuted MNIST**		**Rotated MNIST**		**Split CIFAR-100**	
	Accuracy ↑	Forgetting ↓	Accuracy ↑	Forgetting ↓	Accuracy ↑	Forgetting ↓
Naive SGD	44.4 (±2.46)	0.53 (±0.03)	46.3 (±1.37)	0.52 (±0.01)	40.4 (±2.83)	0.31 (±0.02)
EWC (Kirkpatrick et al., 2017)	70.7 (±1.74)	0.23 (±0.01)	48.5 (±1.24)	0.48 (±0.01)	42.7 (±1.89)	0.28 (±0.03)
A-GEM (Chaudhry et al., 2019a)	65.7 (±0.51)	0.29 (±0.01)	55.3 (±1.47)	0.42 (±0.01)	50.7 (±2.32)	0.19 (±0.04)
ER-Reservoir (Chaudhry et al., 2019b)	72.4 (±0.42)	0.16 (±0.01)	69.2 (±1.10)	0.21 (±0.01)	46.9 (±0.76)	0.21 (±0.03)
Stable SGD (Mirzadeh et al., 2020b)	80.1 (±0.51)	0.09 (±0.01)	70.8 (±0.78)	0.10 (±0.02)	56.9 (±1.52)	0.11 (±0.01)
MC-SGD (Mirzadeh et al., 2021)	82.9 (±0.40)	0.10 (±0.01)	81.9 (±0.46)	0.08 (±0.01)	58.2 (±0.91)	0.08 (±0.01)
Ensemble MC-SGD	88.30 (±0.48)	0.06 (±0.01)	83.63 (±0.39)	0.07 (±0.01)	64.12 (±1.16)	0.06 (±0.01)
Batch Ensemble (Wen et al., 2020)	63.37 (±1.32)	0.27 (±0.01)	57.54 (±0.53)	0.17 (±0.01)	53.08 (±1.70)	0.08 (±0.01)
Subspace-Connectivity (ours)	87.8 (±0.53)	0.06 (±0.01)	86.7 (±0.67)	0.07 (±0.01)	61.7 (±0.80)	0.05 (±0.01)
Multitask Learning	89.5 (±0.21)	0.0	89.8(±0.37)	0.0	66.8(±1.42)	0.0

Table 6: Comparison between the proposed method (Subspace-Connectivity) and other single model baselines on 5 random seeds (± std).

[7] this value must be multiplied by the number of model n to get the final learning rate used. Same logic is applied for miniImageNet dataset.



(a) Permuted MNIST

(b) Rotated MNIST

(c) Split CIFAR-100

Figure 16: Evolution of the average accuracy throughout the training.

Method	**Split miniImageNet**	
	Accuracy ↑	Forgetting ↓
Naive SGD	43.66 (±1.65)	0.22 (±0.02)
ER Reservoir (Chaudhry et al., 2019b)	51.7 (±2.53)	0.11 (±0.02)
Stable SGD (Mirzadeh et al., 2020b)	53.76 (±1.13)	0.07 (±0.01)
MC-SGD (Mirzadeh et al., 2021)	54.80 (±1.04)	0.05 (±0.01)
Ensemble MC-SGD	59.2 (±1.1)	0.04 (±0.01)
Batch Ensemble (Wen et al., 2020)	51.78 (±1.74)	0.05 (±0.01)
Subspace-Connectivity (ours)	58.17 (±0.84)	0.03 (±0.01)
Multitask Learning	62.82 (±1.77)	0.0

Table 7: Comparison between Subspace-Connectivity and other baselines on 5 random seeds for Split miniImageNet.