Representation Ensembling for Synergistic Lifelong Learning with Quasilinear Complexity

Anonymous authors
Paper under double-blind review

Abstract

In lifelong learning, data are used to improve performance not only on the current task, but also on previously encountered, and as yet unencountered tasks. In contrast, classical machine learning which starts from a blank slate, or *tabula rasa*, uses data only for the single task at hand. While typical transfer learning algorithms can improve performance on future tasks, their performance on prior tasks degrades upon learning new tasks (called forgetting). Many recent approaches for continual or lifelong learning have attempted to *maintain* performance on old tasks given new tasks. But striving to avoid forgetting sets the goal unnecessarily low. The goal of lifelong learning should be to not only improve performance on future tasks (forward transfer) but also to improve performance on past tasks (backward transfer) with any new data. Our key insight is that we can synergistically ensemble representations that were learned independently on disparate tasks to enable both forward and backward transfer. This generalizes ensembling independently learned representations (like in decision forests) and complements ensembling dependent representations (like in gradient boosted trees). Moreover, we ensemble representations in quasilinear space and time. We demonstrate this insight with two algorithms: representation ensembles of (1) trees and (2) networks. Both algorithms demonstrate forward and backward transfer in a variety of simulated and benchmark data scenarios, including tabular, image, spoken, and adversarial tasks, including CIFAR-100, Five-Dataset, Split Mini-Imagenet, and Food1k, as well as the spoken digit dataset. This is in stark contrast to the reference algorithms we compared to, most of which failed to transfer either forward or backward, or both, despite that many of them require quadratic space or time complexity.

1 Introduction

Learning is a process by which an intelligent system improves performance on a given task by leveraging data (Mitchell, 1999). In classical machine learning, the system is often optimized for a single task (Vapnik & Chervonenkis, 1971; Valiant, 1984). While it is relatively easy to *simultaneously* optimize for multiple tasks (multi-task learning) (Caruana, 1997), it has proven much more difficult to *sequentially* optimize for multiple tasks (Thrun, 1996; Thrun & Pratt, 2012). Specifically, classical machine learning systems, and natural extensions thereof, exhibit "catastrophic forgetting" when trained sequentially, meaning their performance on the prior tasks drops precipitously upon training on new tasks (McCloskey & Cohen, 1989; McClelland et al., 1995). However, learning could be lifelong, with agents continually building on past knowledge and experiences, improving on many tasks given data associated with any task. For example, learning a second language often improves performance in an individual's native language (Zhao et al., 2016).

In the past 30 years, a number of sequential task learning algorithms have attempted to overcome catastrophic forgetting. These approaches naturally fall into one of two camps. In one, the algorithm has fixed resources, and so must reallocate resources (essentially compressing representations) in order to incorporate new knowledge Kirkpatrick et al. (2017); Zenke et al. (2017); Li & Hoiem (2017); Schwarz et al. (2018); Finn et al. (2019). For efficient compression of the representation, the model weights can be regularized by using information extracted from old tasks or by directly replaying the old task data. Biologically, this corresponds to adulthood, where brains have a nearly fixed or decreasing number of cells and synapses. In the other, the algorithm adds (or builds) resources as new data arrive (essentially ensembling representations) (Ruvolo & Eaton, 2013; Rusu et al., 2016; Lee et al., 2019). Biologically, this corresponds to development, where brains grow by adding cells, synapses, etc. A close resemblance to this resource growing approach can be found in Sodhani et al. (2020), where the model adaptively expands when the capacity of the model saturates.

Approaches from both camps demonstrate some degree of continual (or lifelong) learning (Parisi et al., 2019). In particular, they can sometimes learn new tasks while not catastrophically forgetting old tasks. However, as we will show, many reference lifelong learning algorithms are unable to transfer knowledge forward (to future unseen tasks) and most of them do not transfer backward (to previously seen tasks). With high enough sample sizes, some of them are able to transfer forward or backward, but transfer is more important in low sample size regimes (Chen & Liu, 2016; Lee et al., 2019). This inability to effectively transfer in low-sample size regimes has been identified as one of the key obstacles limiting the capabilities of artificial intelligence (Pearl, 2019; Marcus & Davis, 2019). We focus primarily on the (arguably simpler) resource growing camp in which each new task is learned with additional representational capacity. We consider a batch learning environment within each task (Kirkpatrick et al., 2017; Schwarz et al., 2018; Zenke et al., 2017; Li & Hoiem, 2017; Rusu et al., 2016; Lee et al., 2019), where we know the task identities during training and inference. The tasks are streaming, but the data within the tasks are batched.

Prior work illustrates that ensembling learners can yield huge advantages in a wide range of applications. For example, in classical machine learning, ensembling trees leads to state-of-the-art random forest (Breiman, 2001) and gradient boosting tree algorithms (Chen & Guestrin, 2016). Similarly, ensembling networks shows promising results in various real-world applications (Qiu et al., 2014; Potes et al., 2016). Wang et al. (2003) used weighted ensemble of learners in a streaming setting with distribution shift. `TrAdaBoost` Dai et al. (2007) boosts ensemble of learners to enable transfer learning. In continual learning scenarios, many algorithms have been built on these ideas by ensembling dependent representations. For example, `Learn++` Polikar et al. (2001) boosts ensembles of weak learners learned over different data sequences in class incremental lifelong learning settings van de Ven et al. (2022). `Model Zoo` (Ramesh & Chaudhari, 2021) uses the same boosting approach in task incremental lifelong learning scenarios van de Ven et al. (2022).

Another group of algorithms, `ProgNN` (Rusu et al., 2016) and `DF-CNN` (Lee et al., 2019) learn a new "column" of nodes and edges with each new task, and ensembles the columns for inference (such approaches are commonly called 'modular' now). The primary difference between `ProgNN` and `DF-CNN` is that `ProgNN` has forward connections to the current column from all the past columns. This creates the possibility of forward transfer while freezing backward transfer. However, the forward connections in `ProgNN` render it computationally inefficient for a large number of tasks. `DF-CNN` gets around this problem by learning a common knowledge base and thereby, creating the possibility of backward transfer.

Recently, many modular approaches have been proposed in the literature that improve on `ProgNN`'s capacity growth. These methods consider the capacity for each task being composed of modules that can be shared across tasks and grown as necessary. For example, `PackNet` Mallya & Lazebnik (2018) starts with a fixed capacity network and trains for additional tasks by freeing up portion of the network capacity using iterative pruning. Veniat et al. (2020) trains additional modules with each new task, and the old modules are only used selectively. Ostapenko et al. (2021) improved the memory efficiency of the modular methods by adding new modules according to the complexity of the new tasks. Mehta et al. (2021) proposed non-parametric factorization of the layer weights that promotes sharing of the weights between tasks. However, all of modular methods described above lack backward transfer because the old modules are not updated with the new tasks. Dynamically Expandable Representation (DER) (Yan et al., 2021) proposed an improvement over the modular approaches where the model capacity is dynamically expanded and the model is fine-tuned by replaying a portion of the old task data along with the new task data. This approach achieves backward transfer between tasks as reported by the authors in the experiments.

Another strategy for building lifelong learning machines is to use total or partial replay (van de Ven et al., 2020; Robins, 1995; Shin et al., 2017; van de Ven et al., 2020). Replay approaches keep the old data and replay them when faced with new tasks to mitigate catastrophic forgetting. However, as we will illustrate, previously proposed replay algorithms do not demonstrate positive backward transfer in our experiments, though they often do not forget as much as other approaches.

Our approach builds directly on previously proposed modular and replay approaches with one key distinction: in our approach, representations are learned independently. The conceptual advantage of growing statistically independent representations per task is similar to the conceptual advantage of growing statistically independent representations per tree in random forest: Brieman (Breiman, 2001) showed that doing so asymptotically yields optimal performance. Empirically, for low sample sizes random forests (which learn independent trees) typically outperform gradient boosted trees (which learn dependent trees) Caruana et al. (2004; 2008); Fernández-Delgado et al. (2014). Because transfer is particularly important in the low-sample size regime, we expect learning independent representations to outperform learning dependent representations in these scenarios as well. Moreover, independent representations also have computational advantages, as doing so merely requires quasilinear time and space, and can be learned in parallel. To leverage the independent representations across past and future tasks, we introduce a channel layer which does the ensembling. We empirically find that algorithms which grow capacity like modular approaches as well as replay old task data with additional tasks demonstrate both forward and backward learning capabilities (see `SynN`, `SynF` and `Model Zoo` in Figure 1).

We developed two complementary lifelong learning algorithms, one based on ensembling decision forests (Syngeristic Forests, `SynF`), and another based on ensembling deep networks (Synergistic Networks, `SynN`). The representation learned by both decision forests and deep networks can be characterized by polytopes that partition the feature space (Priebe et al., 2020). `SynF` and `SynN` ensemble sets of polytopes learned from each task by aggregating discriminative information across tasks via a channel layer.

We explore our proposed algorithm as compared to a number of reference algorithms on an extensive suite of numerical experiments that span simulations, vision datasets including CIFAR-100, 5-dataset, Split Mini-Imagenet, and Food1k, as well as the spoken digit dataset. Figure 1 illustrates that our algorithms outperform all the reference algorithms in terms of forward, backward, and overall transfer. This is the case for our proposed resource growing algorithms, using either previously proposed or our novel evaluation criteria. Ablation studies indicate the degree to which the amount of representation or storage capacity and replaying old task data impact performance of our algorithms. All our code and experiments are open source to facilitate reproducibility.

2 Background

2.1 Classical Machine Learning

Classical supervised learning (Mohri et al., 2018) seeks to learn a map from inputs, $x \in \mathcal{X}$ to outputs, $y \in \mathcal{Y}$. To do so, we often begin with a dataset consisting of n input/output pairs, $\mathbf{S}_n = \{(X_i, Y_i)\}_{i=1}^n$. We use those data to learn a hypothesis, $h \in \mathcal{H}$ that maps new inputs, x to the correct output, y. To do so, we assume that each (x, y) pair is a realization of a random variable, (X, Y), and that each pair is sampled identically and independently from some joint distribution, $(X, Y) \overset{iid}{\sim} \mathcal{D}$. Given a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$, the goal is to find the hypothesis, $h : \mathcal{X} \to \mathcal{Y}$ that minimizes *risk*, typically defined as expected loss, $R(h) = \mathbb{E}_{(X,Y)\sim\mathcal{D}}\left[\ell(h(X), Y)\right]$.

A learning algorithm is a function f that maps datasets, $\mathbf{S}_n$, to a hypothesis h_n, and is evaluated on its generalization error (or expected risk): $\mathbb{E}_{\mathbf{S}_n\sim\mathcal{D}^n}\left[R(f(\mathbf{S}_n))\right] = \mathbb{E}\left[R(\hat{h}_n)\right]$, where the expectation is taken with respect to the true but unknown distribution governing the training data, $\mathcal{D}^n$. We use subscript with $\mathcal{D}$ to denote the distribution when we sample a single point and superscript to denote a joint distribution when we simultaneously sample multiple points. The goal is to choose a learner f that learns a hypothesis $\hat{h}_n$ using n training samples that has a small generalization error for the given task (Bickel & Doksum, 2015).

2.2 Multiple tasks in batch and sequential modes

2.2.1 Multitask Learning

Lifelong learning is a generalization of multitask learning, which is itself a generalization of classical machine learning described above. We therefore first explain multitask learning. Multitask supervised learning

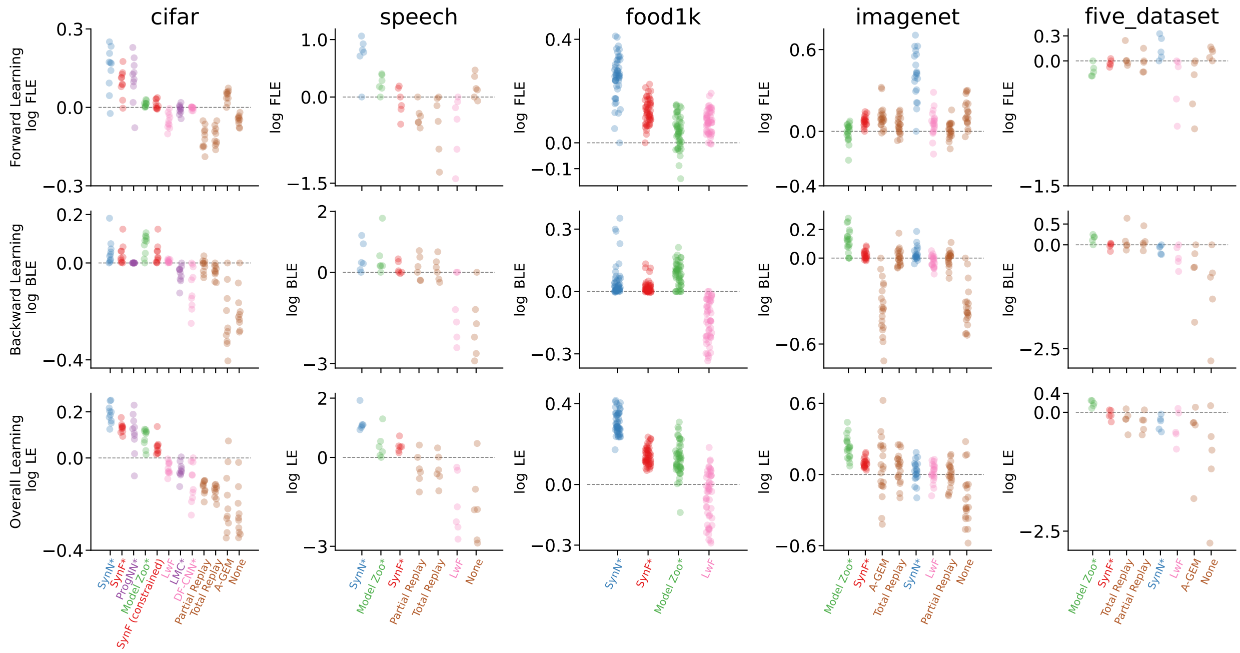

Figure 1: **Performance summary on vision and audition benchmark datasets.** `SynN` and `SynF` shows positive forward and backward transfer on CIFAR 10X10 (row 1), Speech (row 2), FOOD1k (row 3), 5-dataset (row 4), Split Mini-Imagenet (row 5) datasets. The datasets are sorted according to the ascending order of the number of samples per task from left to right. The algorithms for each dataset on the x-axis is sorted according to the descending order of the mean log LE for each algorithm. To differentiate between resource growing and resource constrained algorithms, a '*' has been added to the end of the name of the resource growing algorithms along the x-axis. As `EWC`, `O-EWC`, `SI`, `TAG` and `ER` always perform worse than `LwF`, we do not show them in the summary plot.

generalizes classical supervised machine learning in two new ways: (i) instead of one task, there is an environment $\mathcal{T}$ of (possibly infinitely) many tasks, and (ii) datasets $\mathbf{S}^t = \{(X_i, Y_i)\}_{i=1}^{n_t}$ where $(X, Y) \overset{iid}{\sim} \mathcal{D}_t$. In supervised learning settings, one can consider the following risk for a particular task t with n random training samples $\mathbf{S}^t$:

$$R^t(f(\mathbf{S}^t)) = R^t(\hat{h}_n) = \mathbb{E}_{(X,Y)\sim\mathcal{D}_t}[\ell(\hat{h}_n(X), Y)]. \tag{1}$$

Note that the data $\mathbf{S}^t$ is a random variable distributed as $\mathcal{D}^t$ and it may contain data that is relevant to any number of tasks (potentially all the tasks) in the environment. One may take expectation with respect to $\mathcal{D}^t$ for averaging out the randomness in the risk due to $\mathbf{S}^t$ and consider the generalization error for the task as:

$$\mathcal{E}_f^t(\mathbf{S}^t) = \mathbb{E}_{\mathbf{S}^t\sim\mathcal{D}^t}[R^t(f(\mathbf{S}^t))]. \tag{2}$$

We can also aggregate the T datasets to obtain a single dataset, $\mathbf{S}_n = \cup_{t=1}^{T}\mathbf{S}^t$ distributed as $\mathcal{D}^n$, and feed those data into f (assuming the input/output spaces are suitably defined.) Then, we can consider the overall generalization error as follows. We are given the error $\mathcal{E}_f^t(\mathbf{S}_n)$ for $t = 1, \cdots, T$ and a weight for each task m_t corresponding to the extent the learner prioritizes task t such that $\sum_{t=1}^{T} m_t = 1$ and $m_t \geq 0$. For example, consider a biological learner which may prioritize the tasks necessary for its survival more than other tasks. However, we use equal priority for all the tasks in this paper which is a common practice in the literature Rusu et al. (2016); Lee et al. (2019); Ramesh & Chaudhari (2021). Letting $\mathcal{E}_f^{\mathcal{T}}(\mathbf{S}_n) = \sum_{t\in\mathcal{T}} m_t \; \mathcal{E}_f^t(\mathbf{S}_n)$ and given a class of learners $\mathcal{F}$, the goal of a multi-task learner is to find an $f \in \mathcal{F}$ such that:

$$\begin{array}{ll} \text{minimize} & \mathcal{E}_f^{\mathcal{T}}(\mathbf{S}_n) \\ \text{subject to} & f \in \mathcal{F} \end{array} . \tag{3}$$

2.2.2 Lifelong Learning

A multitask learner has access to all the data at the same time. Lifelong learning generalized multitask learning by assuming the data per task, and/or the tasks, arrive sequentially. To avoid the possibility that the learner merely stores all the old data and recomputes everything from scratch, lifelong learning problems also impose computational and storage complexity constraints.

Given T tasks so far, a current hypothesis $\hat{h}_m$ trained on data $\mathbf{S}_m$, a lifelong learning algorithm f takes a new dataset, $\mathbf{S}_{n'}$, and uses it to update the hypothesis, $f(\hat{h}_m, \mathbf{S}_{n'}) \mapsto \hat{h}_n$, where $n = n' + m$, to solve Equation equation 3. Implicit in Equation equation 3 is that we are concerned not only with past tasks $t < T$, but also all possible future tasks, $t \in \mathcal{T}$. Directly solving Equation equation 3 is not possible because it requires knowing the exact distribution for each task, therefore, we have devised indirect ways of solving it.

The computational complexity constraints for lifelong learning are crucial, though often implicit. For example, consider the algorithm that stores all the data, and then retrains everything from scratch each time a new sample arrives. Without computational constraints, such an algorithm could be classified as a lifelong learner; we do not think such a label is appropriate for that algorithm. Thus, we only consider learners f lifelong learners assuming their performance scales sub-quadratically with sample size (see below for details). The goal in lifelong learning therefore is, given new data and a new task, use all the existing data to achieve lower generalization error on this new task, while also using the new data to obtain a lower generalization error on the previous tasks. This is distinct from classical online learning scenarios (Cesa-Bianchi & Lugosi, 2006), because the previously experienced tasks may recur, so we are concerned about maintaining and improving performance on those tasks as well. In "task-aware" scenarios, the learner or hypothesis is aware of all task details for all tasks. In "task-unaware" (or agnostic (Zeno et al., 2018)) scenarios the learner or hypothesis does not know that the task has changed at all. We only address task-aware scenarios here.

2.3 Reference algorithms

We compared our approaches to 15 reference lifelong learning methods. These algorithms can be classified into two groups based on whether they add capacity resources per task, or not.

Among them, `ProgNN` (Rusu et al., 2016), Deconvolution-Factorized CNNs (`DF-CNN`) (Lee et al., 2019) and `LMC` (Ostapenko et al., 2021) learn new tasks by building new resources. For `ProgNN`, for each new task a new "column" of network is introduced. In addition to introducing this column, lateral connections from all previous columns to the new column are added. These lateral connections are computationally costly, as explained in Subsection 4.2. `DF-CNN` (Lee et al., 2019) is a lifelong learning algorithm that improves upon `ProgNN` by introducing a knowledge base with lateral connections to each new column, thereby avoiding all pairwise connections, and dramatically reducing computational costs. `LMC` improves further upon `ProgNN` and `DF-CNN` by introducing new layers or module only when a sufficient deviation from the previous tasks is detected in the locally decomposed modules.

We also compare two variants of exact replay (Total Replay and Partial Replay) using the code provided by van de Ven et al. (2020). Both Total and Partial Replay store all the data they have ever seen, but Total Replay replays all of it upon acquiring a new task, whereas Partial Replay replays M samples, randomly sampled from the entire corpus, whenever we acquire a new task with M samples. We have also compared our approach with more constrained ways of replaying old task data, including `Model Zoo`(Ramesh & Chaudhari, 2021), Averaged Gradient Episodic Memory (`A-GEM`) (Chaudhry et al., 2018), Experience Replay (`ER`) (Chaudhry et al., 2019) and Task-based Accumulated Gradients (`TAG`) (Malviya et al., 2021). Among them, `Model Zoo` ensembles multiple representations using the boosting approach. In `Model Zoo`, the total number of networks within the ensemble was capped at the total number of tasks to make it comparable with our approach. For `A-GEM` and `ER`, the size of episodic memory is set to store 1 example per class. On the other hand, TAG stores the gradients or directions the model took while learning a specific task instead of storing past examples.

The other five algorithms, Elastic Weight Consolidation (`EWC`) (Kirkpatrick et al., 2017), Online-EWC (`O-EWC`) (Schwarz et al., 2018), Synaptic Intelligence (`SI`) (Zenke et al., 2017), Learning without Forgetting (`LwF`) (Li & Hoiem, 2017), and "None," all have fixed capacity resources. For the baseline "None", the network was incrementally trained on all tasks in the standard way while always only using the data from the current task. The implementations for all of the algorithms are adapted from open source codes (Lee et al., 2019; van de Ven & Tolias, 2019); for implementation details, see Appendix D.

3 Evaluation Criteria

Others have previously introduced criteria to evaluate transfer, including forward and backward transfer (Lopez-Paz & Ranzato, 2017; Benavides-Prado et al., 2018; Díaz-Rodríguez et al., 2018; Veniat et al., 2020). Pearl Judea (2018) introduced the transfer benefit ratio, which builds directly off relative efficiency from classical statistics (Bickel & Doksum, 2015). We refer to these and other such criteria as statistics that quantify lifelong learning performance (rather than metrics, which formally means a notion of distance). As always, no single statistic can serve all purposes because it is a scalar summary of a corpus of data. Below, we illustrate scenarios in which existing metrics fail to provide the kinds of insights that we desire, therefore motivating us to clearly enumerate our desired properties. We then describe our proposed statistics, and finally illustrate how they resolve the issues.

3.1 Failure modes of existing lifelong learning statistics

Consider the simplest lifelong learning scenario, with two tasks. We train our lifelong learner sequentially on Task 1 and then Task 2. Imagine that there is no transfer of learning between the tasks. In other words, assume that training on both tasks does not improve accuracy on either task, relative to only training on each task individually. Further imagine error using this algorithm with this amount of data is lower for Task 2 than for Task 1. Thus, plotting average accuracy over the tasks as a function of the number of tasks would indicate that accuracy is *increasing.* Using accuracy as a statistic to quantify lifelong learning in this scenario, therefore, would yield a false positive. Now imagine that there is forward transfer from Task 1 to Task 2. In other words, Task 2 performance is better upon pre-training on Task 1. And further assume that the error using this algorithm with this amount of data is higher on Task 2 than Task 1. Thus, plotting accuracy as a function of the number of tasks would indicate that accuracy is *decreasing.* Using accuracy as a statistic to quantify lifelong learning in this scenario, therefore, would yield a *false negative.*

Now, imagine that we do transfer from Task 1 to Task 2. But, performance on Task 2 is quite poor even with transfer. Specifically, the accuracy increases from 51% to 52%. The difference in accuracies is 1%. Consider a different scenario where the accuracy on Task 2 without pre-training is already quite high, like 98%. Now, upon pre-training with Task 1, accuracy gets up to 99%. And, the difference in accuracy is still only 1%. However, the later scenario is relatively harder to achieve. This is because as the learner achieves performance closer to Bayes optimum, its improvement with more training samples gets negligible. Therefore, transfer learning is "harder" in higher accuracy regimes compared to that in lower accuracy regimes. We desire statistics that differentiate between the above two distinct cases of improving by 1%.

Forward transfer, as defined by Lopez-Paz & Ranzato (2017) and modified by Díaz-Rodríguez et al. (2018), is defined by the increase in accuracy from all the past task data the learner has seen, but without any of the current task data (so it is effectively quantifying meta-learning or zero-shot learning of the new task given old task data). On the contrary, backward transfer results from the future task data and is defined as the difference in accuracy at different stages of learning of the learner. Backward transfer as defined by them is actually a different function from forward transfer, rather than the same function with different inputs.

Veniat et al. (2020) modifies and extends the definitions from Lopez-Paz & Ranzato (2017) and Díaz-Rodríguez et al. (2018) They ignore forward transfer, and simply define 'transfer', which is essentially the performance gained in one task using all other tasks, therefore integrating both forward and backward transfer. Their 'forgetting' is essentially the same as Lopez-Paz & Ranzato (2017) backward transfer. Thus, Veniat et al. (2020) has introduced a third different function.

These limitations motivate the construction of statistics that build on accuracy, forward, and backward transfer as defined by others, to address these concerns.

3.2 Desiderata

Here we describe our desiderata, which are designed to move from the specific failure modes illustrated above, to general principles. While there is no universal set of desiderata, we believe this set is sufficient to ascertain which statistics will be insightful about lifelong learning.

1. A *general* definition of learning efficiency which we can apply in multiple distinct scenarios, including generic transfer, and also both forward and backward transfer. We wish to avoid the cognitive load associated with having to remember multiple distinct functions; we prefer one general function that can be applied to different scenarios.

2. Explicitly quantify *transfer*, as opposed to accuracy. It is possible for accuracy for a given learner to increase with more tasks without transfer, or decrease with transfer. We desire to avoid such false positives and negatives.

3. The criteria *normalizes* accuracy. This avoids the issues associated with small absolute changes sometimes being easier, and sometimes being harder to achieve.

4. The total amount of transfer naturally *decomposes* in forward and backward transfer. This simplifies understanding the contribution of various aspects of transfer.

3.3 Our Performance Measure

We are interested in measuring the ratio of the generalization error of an algorithm that has learned on one dataset, as compared to the generalization error of that same algorithm on a different dataset. Typically, we are interested in situations where the former dataset is a subset of the latter dataset. For example, we define transfer efficiency as:

Definition 1 (Transfer Efficiency) *The transfer efficiency* (TE) *for task t with learner f is defined as:*

$$\mathsf{TE}_f^t(\mathbf{S}, \mathbf{S}') = \frac{\mathcal{E}_f^t(\mathbf{S})}{\mathcal{E}_f^t(\mathbf{S}')},$$

where each expectation in the error is taken with respect to the corresponding data set distribution, i.e. the numerator expectation is taken with respect to the distribution governing $\mathbf{S}$, and the denominator expectation is taken with respect to the distribution governing $\mathbf{S}'$.

As we will show below, Definition 1 satisfies Desiderata 1. Applying this definition for situations with multiple tasks (whether the tasks arrive in batch or sequentially), we derive learning efficiency, forward and backward learning efficiency.

Definition 2 (Learning Efficiency) *The learning efficiency of algorithm f for given Task t with total sample size n is:*

$$\mathsf{LE}_n^t(f) := \frac{\mathcal{E}_f^t(\mathbf{S}^t)}{\mathcal{E}_f^t(\bigcup_{t'=1}^{T} \mathbf{S}^{t'})}. \tag{4}$$

We say that algorithm f has transferred across all the tasks up to T with data $\mathbf{S}$ if and only if $\mathsf{LE}_n^t(f) > 1$ for all the tasks up to T.

To evaluate a lifelong learning algorithm while respecting the streaming nature of the tasks, it is convenient to consider two extensions of learning efficiency. Overall learning efficiency satisfies Desiderata 2 and 3 because it is a ratio of performances of a learner with and without additional data, thereby quantifying transfer relative to the learner without additional data. *Forward* learning efficiency is the expected ratio of the generalization error of the learning algorithm with (i) access only to Task t data, (ii) access to the data up to and including the last observation from Task t. This quantity measures the relative effect of previously seen out-of-task data on the performance on Task t.

Definition 3 (Forward Learning Efficiency) *The forward learning efficiency of f for task t given n samples is :*

$$\mathsf{FLE}_n^t(f) := \frac{\mathcal{E}_f^t(\mathbf{S}^t)}{\mathcal{E}_f^t(\bigcup_{t'=1}^{t} \mathbf{S}^{t'})}. \tag{5}$$

We say an algorithm (positively) forward transfers for task t if and only if $\mathsf{FLE}_n^t(f) > 1$. In other words, if $\mathsf{FLE}_n^t(f) > 1$, then the algorithm has used data associated with past tasks to improve performance on task t. Note that a learner has only forward transfer from the past tasks to a specific task only when the task is introduced to the learner.

One can also determine the rate of *backward* transfer by comparing the generalization error $\mathcal{E}_f^t(\bigcup_{t'=1}^{t} \mathbf{S}^{t'})$ to the generalization error of the hypothesis learned having seen the entire training dataset up to Task T. More formally, backward learning efficiency is the ratio of the generalization error of the learned hypothesis with (i) access to the data up to and including the last observation from task t, to (ii) access to the entire dataset. Thus, this quantity measures the relative effect of future task data on the performance on Task t.

Definition 4 (Backward Learning Efficiency) *The backward learning efficiency of f for Task t given n samples is*

$$\mathsf{BLE}_n^t(f) := \frac{\mathcal{E}_f^t(\bigcup_{t'=1}^{t} \mathbf{S}^{t'})}{\mathcal{E}_f^t(\bigcup_{t'=1}^{T} \mathbf{S}^{t'})}. \tag{6}$$

We say an algorithm (positively) backward transfers to Task t from all the tasks T if and only if $\mathsf{BLE}_n^t(f) > 1$. We can report $\mathsf{BLE}_n^t(f)$ for each t as we gradually increase the number of total task T in the environment or we can report the final $\mathsf{BLE}_n^t(f)$ for each t after we are done adding task to the environment as a summary. The former measure shows the dynamics of the task specific performance whereas the latter one shows an average performance from all the tasks. In summary, if $\mathsf{BLE}_n^t(f) > 1$, then the algorithm has used data associated with future tasks to improve performance on past tasks.

Our definitions of forward and backward transfer efficiency indicate that learning efficiency also satisfies Desiderata 1, as the same equation is used for general, forward, and backward learning efficiency.

After observing T tasks, the extent to which the LE for the t^{th} task comes from forward transfer versus from backward transfer depends on the order of the tasks. If we have a sequence in which tasks do not repeat, learning efficiency for the first task is all backward transfer, for the last task it is all forward transfer, and for the middle tasks it is a combination of the two. In general, LE factorizes into FLE and BLE:

$$\mathsf{LE}_n^t(f) = \frac{\mathcal{E}_f^t(\mathbf{S}^t)}{\mathcal{E}_f^t(\bigcup_{t'=1}^{T} \mathbf{S}^{t'})} = \frac{\mathcal{E}_f^t(\mathbf{S}^t)}{\mathcal{E}_f^t(\bigcup_{t'=1}^{t} \mathbf{S}^{t'})} \times \frac{\mathcal{E}_f^t(\bigcup_{t'=1}^{t} \mathbf{S}^{t'})}{\mathcal{E}_f^t(\bigcup_{t'=1}^{T} \mathbf{S}^{t'})}. \tag{7}$$

The above equation indicates that our proposed quantity also satisfies Desiderata 4. Of note, previously proposed quantities for evaluating lifelong learning performance could likely also be decomposed as we have done, though we have not seen it in the literature.

Throughout, we will report log LE so that positive learning corresponds to $\mathsf{LE} > 1$. In a lifelong learning environment having T tasks drawn with replacement from $\mathcal{T}$, learner f $\boldsymbol{m}$-lifelong learns tasks $t \in \mathcal{T}$ if the log of the convex combination of learning efficiencies is greater than 0, that is,

$$\log \sum_{t \in \mathcal{T}} m_t \cdot \mathsf{LE}_n^t(f) > 0 \tag{8}$$

where m_t corresponds to the extent to which the learner prioritizes a certain task t. Note that when m_t is equal for each task, the learner has to excel equally in each task. We say an agent has **synergistically learned** in an environment of T tasks if the agent has positively learned, i.e., the left hand side of Inequality equation 8 is positive for all of the possible convex combinations of all the tasks up to T.

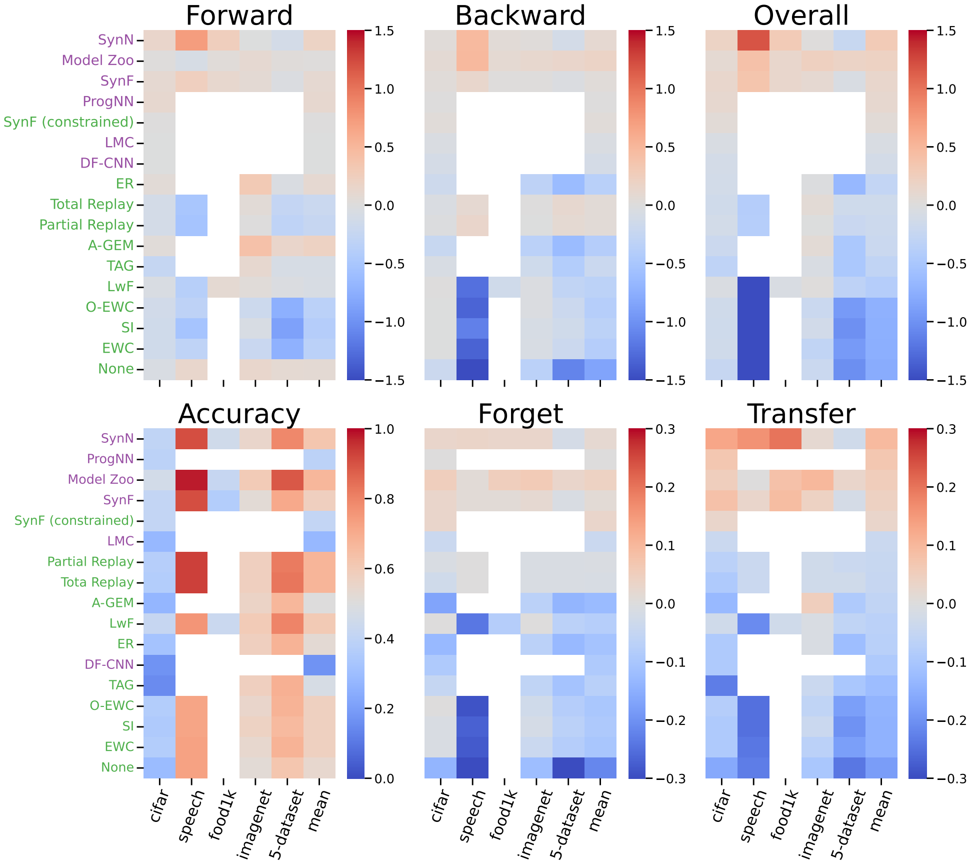

Figure 2: **Performance summary on vision and audition benchmark datasets using Veniat's and our proposed statistics.** The empty row for a benchmark data represents the algorithms that we did not or could not run using the code provided by the authors. The last column in each panel includes the mean performance over all the datasets. *Top:* Average $\log \mathsf{FLE}$ (left) , $\log \mathsf{BLE}$ (middle) and $\log \mathsf{LE}$ (right) over all the tasks for different datasets after the final task has been introduced. The right panel is summation of the left two panels. *Bottom:* Average accuracy (left), forgetting (middle) and average transfer (right) as proposed by Veniat et al. (2020). Resource growing and resource constrained algorithms are color coded as purple and green, respectively. The order of the algorithms is sorted according to the decreasing sequence of the last column (mean) of the right panel (top and bottom). Our proposed statistics provide intuitive ordering of the lifelong learning agents according to their performance.

3.4 Do LE, FLE, BLE mitigate the above described failure modes?

As shown above in the definition, our proposed statistics satisfy desiderata 1. Moreover, LE and FLE quantify the improvement of a lifelong learner in comparison with its single task counterparts. Thus, they explicitly quantify transfer as opposed to a mere improvement in accuracy which satisfies desiderata 2. Now,

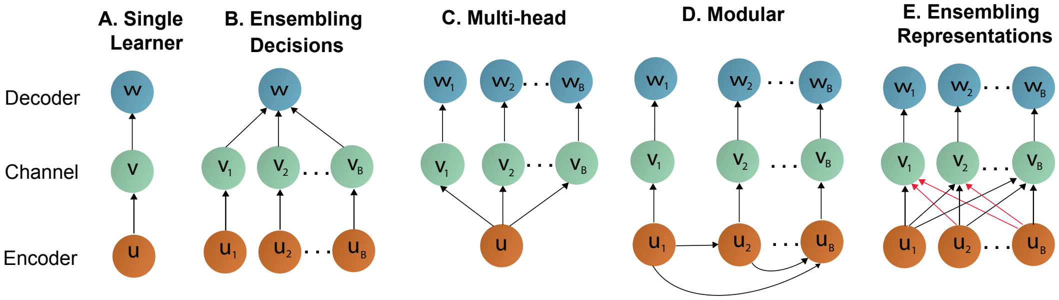

Figure 3: Schemas of composable hypotheses. A. Single task learner. B. Ensembling decisions (as output by the channels) is a well-established practice, including random forests and gradient boosted trees. C. Learning a joint representation or D. Ensembling representations (learned by the encoders) was previously used in lifelong learning scenarios, but were not trained independently as in E, thereby causing interference or forgetting. Note that the new encoders interact with the previous encoders through the channel layer (indicated by red arrows), thereby, enabling backward transfer. Again the old encoders interact with the future encoders (indicated by black arrows), thereby, enabling forward transfer.

consider the two distinct cases of 1% accuracy improvement as mentioned before in Section 3.1. LE for the two scenarios are 2 and 1.02, respectively. Thus, our proposed statistics normalizes the relative hardship in transfer depending on the level of accuracy. However, we approximate risk, as defined in Equation equation 2, via several Monte Carlo. Thus, the estimated risk may be noisy. In practice, this does not seem to be an issue.

To further elucidate the utility of our proposed performance statistics, we report the the statistics closely related and recently proposed by Veniat et al. (2020) along with our proposed statistics on the vision and the language benchmark datasets used in this paper in Figure 2. The algorithms in Figure 2 are ordered according to the decreasing order of the mean transfer over all the datasets (last column right panel of Figure 2. Veniat et al. (2020) defined *forget* and *transfer* in a similar way to that of our BLE and LE, respectively. However, *forget* and *transfer* consider the difference in accuracy rather than ratio. Note that in Figure 2, the top right panel is the sum of the top left two panels. Although, the bottom right panel in Figure 2 could be decomposed into forward or backward components, the authors did not mention this. As apparent from the top panel of Figure 2, some of the reference algorithms like ProgNN, DF-CNN, ER, A-GEM, None have high initial accuracy, i.e., forward transfer for each task. However, they eventually forget reflected as poor backward transfer in the top middle panel of Figure 2.

Moreover, the ordering of the algorithms is different in both of the figures. For example, ProgNN has higher transfer than Model Zoo, but significantly lower accuracy. Therefore, Veniat et al. (2020)'s transfer is capturing mostly gains in accuracy, whereas our transfer has normalized for the effect of accuracy. The result of this is that using our statistics Model Zoois performing better than ProgNN (Figure 2), whereas Veniat et al. (2020)'s statistics suggest that ProgNN is outperforming Model Zoo. Another difference is evident upon comparing Partial Replay to Total Replay. Intuitively, Partial Replay should perform worse than Total Replay (van de Ven et al., 2020) because it is subsampling past tasks, rather than using all data to replay each task. However, Veniat et al. (2020)'s statistics suggest that Partial Replay performs better than Total Replay, whereas our statistics provide the more intuitive result.

4 Representation ensembling algorithms

In this section, we provide an abstract idea of our approach and we refine the details of the algorithms further in Subsection 4.1.2 and 4.1.1. We start by leveraging a decomposition of a learner originally proposed by Shannon; a learner has three components: an encoder, a channel, and a decoder (Cover & Thomas, 2012; Cho et al., 2014): $h(\cdot) = w \circ v \circ u(\cdot)$. Figure 3 shows these three components as the building blocks of different learning schemas. The encoder, $u : \mathcal{X} \mapsto \tilde{\mathcal{X}}$, maps an $\mathcal{X}$-valued input into an internal representation space

$\tilde{\mathcal{X}}$ (Vaswani et al., 2017; Devlin et al., 2018). The channel $v : \tilde{\mathcal{X}} \mapsto \Delta_{\mathcal{Y}}$ maps the transformed data into a posterior distribution (or, more generally, a score). Finally, a decoder $w : \Delta_{\mathcal{Y}} \mapsto \mathcal{Y}$, produces a predicted label.

A canonical example of a single learner depicted in Figure 3A is a decision tree. Importantly, we can leverage different data to learn different components of the tree (Breiman et al., 1984; Denil et al., 2014; Athey et al., 2019). For example, we can use some data to learn the tree structure (which is the encoder). Then, by pushing the remaining data (sometimes called the 'out-of-bag' data) through the tree, we can learn posteriors in each leaf node (which are the channel). The channel thus gives scores for each data point denoting the probability of that data point belonging to a specific class. Using separate sets of data to learn the encoder and the channel results in less bias in the estimated posterior in the channels as in 'honest trees' (Breiman et al., 1984; Denil et al., 2014; Athey et al., 2019). Finally, the decoder provides the predicted class label using $\arg\max$ over the posteriors from the channel. See Appendix A for a more detailed and concrete example using a decision tree.

One can generalize the above decomposition by allowing for multiple encoders, as shown in Figure 3B. Given B different encoders, one can attach a single channel to each encoder, yielding B different channels. Doing so requires generalizing the definition of a decoder so that it would operate on multiple channels. Such a decoder ensembles the *decisions*, because here each channel provides the final output based on the encoder. This is the learning paradigm behind bagging (Breiman, 1996) and boosting (Freund, 1995); indeed, decision forests are a canonical example of a decision function operating on an ensemble of B outputs (Breiman, 2001). A decision forest learns B different decision trees, each of which has a tree structure corresponding to an encoder. Each tree is assigned a channel that outputs each tree's vote that an observation is in any class. The decoder outputs the most likely class averaged over the trees.

Although the task specific structure in Figure 3B can provide useful decision on the corresponding task, they can not, in general, provide meaningful decisions on other tasks because those tasks might have completely different class labels. Therefore, in the multi-head structure (Figure 3C) a single encoder is used to learn a joint representation from all the tasks, and a separate channel is learned for each task to get the score or class conditional posteriors for each task, which is followed by each task specific decider (Kirkpatrick et al., 2017; Schwarz et al., 2018; Zenke et al., 2017).

Further modification of the multi-head structure allows ProgNN or other modular approaches to learn separate encoder for each task with forward connections from the past encoders to the current one (Figure 3D). This creates the possibility of having forward transfer while freezing backward transfer. Note that if the encoders are learned independently across different tasks, they may have learned useful *representations* that the tasks can mutually leverage.

Our approach is based on the learning scheme in Figure 3E. This scheme requires generalizing the definition of a channel so that it can operate on multiple encoders. The result is that the channels **ensemble representations** (learned by the encoders), rather than decisions (learned by the channels) as in Figure 3B. In this scenario, generalizing with bagging and boosting, the ensemble of channels then feeds into *each* task specific decoder. When each encoder has learned complementary representations, this representation ensembling approach has certain appealing properties, particularly in multiple task scenarios, including lifelong learning.

4.1 Our representation ensembling algorithms

We have developed two different representation ensembling algorithms based on bagging of the encoders which are trained on different tasks. In both algorithms, as new data from a new task arrives, the algorithm first builds a new independent encoder. Then, it builds the channel for this new task by pushing the new task data through all existing encoders. Thus the channel integrates information across all existing encoders using the new task data, thereby enabling forward transfer. At the same time, if it stores old task data, it can push old task data through the new encoders to update the channels from the old tasks, thereby enabling backward transfer. In either case, new test data are passed through all existing encoders and corresponding channels to make a prediction. As we will show empirically, our two ensemble methods achieve both forward and backward transfer on several benchmark datasets. [1]

The key to both of our algorithms is the realization that both forests and networks partition feature space into a union of polytopes (Priebe et al., 2020). Thus, the internal representation learned by each can be considered a sparse vector encoding which polytope a given sample resides in. We can combine the discriminative information over different sets of polytopes learned over different tasks by populating the polytopes with the corresponding task data and thereby, learn a channel for that specific task (see Appendix A, B and C for detailed description of the proposed approach).

4.1.1 Synergistic Networks

A Synergistic Network (`SynN`) ensembles deep networks. For each task, the encoder u_t in an `SynN` is the "backbone" of a deep network (`DN`), including all but the final layer. Thus, each u_t maps an element of $\mathcal{X}$ to an element of $\mathbb{R}^d$, where d is the number of neurons in the penultimate layer of the `DN`. The channels are learned via k-Nearest Neighbors (k-NN) (Stone, 1977) over the d dimensional representations of $\mathcal{X}$. Recall that a k-NN, with k chosen such that as the number of samples goes to infinity, k also goes to infinity, while $\frac{k}{n} \to 0$, is a universally consistent classifier (Stone, 1977). We use $k = 16 \log_2 n$, which satisfies these conditions.The decoder is the same as above.

`SynN` differs from ProgNN in two key ways. First, recall that ProgNN builds a new neural network "column" for each new task, and also builds lateral connections between the new column and all previous columns. *In contrast,* `SynN` *excludes those lateral connections, thereby greatly reducing the number of parameters and train time.* Moreover, this makes each representation independent, thereby potentially avoiding interference across representations. Second, for inference on task j data, assuming we have observed tasks up to $J > j$, `ProgNN` only leverages representations learned from tasks up to j, thereby excluding tasks $j+1, \ldots, J$. *In contrast,* `SynN` *leverages representations from all J tasks, a key difference which enables backward transfer.*

4.1.2 Synergistic Forests

Synergistic Forests (`SynF`) ensemble decision trees or forests. For each task, the encoder u_t of a `SynF` is the representation learned by a decision forest (Amit & Geman, 1997; Breiman, 2001). The leaf nodes of each decision forest partition the input space $\mathcal{X}$ into polytopes (Breiman et al., 1984). The channel then learns the class-conditional posteriors by populating the polytopes with out-of-bag samples and taking class votes, as in "honest trees" (Breiman et al., 1984; Denil et al., 2014; Athey et al., 2019). Each channel outputs the posteriors averaged across the collection of forests learned over different tasks. The decoder w_t outputs the argmax to produce a single prediction. Recall that honest decision forests are universally consistent classifiers and regressors (Athey et al., 2019), meaning that with sufficiently large sample sizes, under suitable though general assumptions, they will converge to minimum risk. Thus, the single task version of this approach simplifies to an approach called "Uncertainty Forests" (Mehta et al., 2019). Table 1 in the appendix lists the hyperparameters used in the CIFAR experiments.

Note that the amount of additional representation capacity added per task by `SynF` is a function of the amount and complexity of the data for a new task. Contrast this with `SynN` and other deep net based modular or representation ensembling approaches, which *a priori* choose how much additional representation to add, prior to seeing all the new task data. So, `SynF` has capacity, space complexity, and time complexity scale with the complexity and sample size of each task. In contrast, `ProgNN`, `SynN` (and others like it) have a fixed capacity for each task, even if the tasks have very different sample sizes and complexities.

---

[1] Wyner et al. (2017) shows that both bagging and boosting asymptotically converge to the Bayes optimal solution. However, for finite sample size and similar model complexity, we empirically find bagging approach to lifelong learning performs better than that of boosting when the training sample size is low (see Figure 5) whereas boosting performs better on large training sample size (See main text Figure 9 and Appendix Figure 4). This is consistent with similar results in single task learning Caruana & Niculescu-Mizil (2006); Caruana et al. (2004); Díaz-Rodríguez et al. (2018)

4.2 A computational taxonomy of lifelong learning

Lifelong learning approaches can be divided into those with fixed computational space resources, and those with growing space resources. We, therefore, quantify the computational space and time complexities of the internal representation of a number of algorithms. The space complexity of the learner refers to the amount of memory space needed to train the learner (Kuo & Zuo, 2003). We also study the representation capacity of these algorithms. Capacity is defined as the size of the subset of hypotheses that is achievable by the learning algorithm (Zhang et al., 2021).

We use the soft-O notation $\tilde{\mathcal{O}}$ to quantify complexity (van Rooij et al., 2019). Letting n be the sample size and T be the number of tasks, we write that the capacity, space or time complexity of a lifelong learning algorithm is $f(n,t) = \tilde{\mathcal{O}}(g(n,T))$ when $|f|$ is bounded above asymptotically by a function g of n and T up to a constant factor and polylogarithmic terms. For simplifying the calculation, we make the following assumptions:

1. Each task has the same number of training samples.
2. Capacity grows linearly with the number of trainable parameters in the model.
3. The number of epochs is fixed for each task, independent of sample size.
4. For the algorithms with dynamically expanding capacity, we assume the worst case scenario where an equal amount of capacity is added to the hypothesis with an additional task.

Assumption 3 enables us to write time complexity as a function of the sample size. Table 1 summarizes the capacity, space and time complexity of several reference algorithms, as well as our `SynN` and `SynF`. For space and time complexity, the table shows results as a function of n and T, as well as the common scenario where sample size per task is fixed and therefore proportional to the number of tasks, $n \propto T$. For detailed calculation of time complexity see Appendix E.

Table 1: Capacity, space, and time complexity of the representation learned by various lifelong learning algorithms. We show soft-O notation ($\tilde{\mathcal{O}}(\cdot,\cdot)$ defined in main text) as a function of $n = \sum_t^T n_t$ and T, as well as the common setting where n is proportional to T. The bottom three rows show algorithms whose space and time both grow quasilinearly with capacity growing.

Parametric	Capacity	Space		Time		Examples
	(n, T)	(n, T)	$(n \propto T)$	(n, T)	$(n \propto T)$	
parametric	1	1	1	n	n	`O-EWC`, `SI`, `LwF`
parametric	1	T	n	nT	n^2	`EWC`
parametric	1	n	n	nT	n^2	`Total Replay`
semi-parametric	T	T^2	n^2	nT	n^2	`ProgNN`
semi-parametric	T	T	n	n	n	`DF-CNN`
semi-parametric	T	$T+n$	n	n	n	`SynN`, `Model Zoo`, `DER`, `LMC`
non-parametric	n	n	n	n	n	`SynF`, `IBP-WF`

Parametric lifelong learning methods have a representational capacity which is invariant to sample size and task number. Although the space complexity of some of these algorithms grow (because the number of the constraints grows stored by the algorithms grows, or they continue to store more data), their capacity is fixed. Thus, given a sufficiently large number of tasks, in general, eventually all parametric methods will catastrophically forget. `EWC` (Kirkpatrick et al., 2017), `Online EWC` (Schwarz et al., 2018), `SI` (Zenke et al., 2017), and `LwF` (Li & Hoiem, 2017) are all examples of parametric lifelong learning algorithms.

Semi-parametric algorithms' representational capacity grows slower than sample size. For example, if T is increasing slower than n (e.g., $T \propto \log n$), then algorithms whose capacity is proportional to T are semi-parametric. `ProgNN` (Rusu et al., 2016) is semi-parametric, nonetheless, its space complexity $\tilde{\mathcal{O}}(T^2)$ due to the lateral connections. Moreover, the time complexity for `ProgNN` also scales quadratically with n when $n \propto T$. Thus, an algorithm that literally stores all the data it has ever seen, and retrains a fixed size network on all those data with the arrival of each new task, would have smaller space complexity and the same time complexity as `ProgNN`. For comparison, we implement such an algorithm and refer to it as Total Replay. `DF-CNN` (Lee et al., 2019) improves upon `ProgNN` by introducing a "knowledge base" with lateral connections to each new column, thereby avoiding all pairwise connections. Because these semi-parametric methods have a fixed representational capacity per task, they will either lack the representation capacity to perform well given sufficiently complex tasks, and/or will waste resources for very simple tasks. `SynN` and `SynF` eliminate the lateral connections between columns of the network, thereby reducing space complexity down to $\tilde{\mathcal{O}}(T)$. They store all the data to enable backward transfer, but retain linear time complexity. Because the time required for pushing the old task data though the old encoders and learning or updating channels is negligible in comparison with the time required for training a new encoder.

Non-parametric algorithms' representational capacity grows linearly with sample size. `SynF` is a non-parametric lifelong learning algorithm with its capacity, space and time complexity all $\tilde{\mathcal{O}}(n)$, meaning that its representational capacity naturally increases with the complexity of each task. Apart from `SynF`, Indian Buffet Process for Weight Factors (`IBP-WF`) (Mehta et al., 2021) proposed the only other non-parametric lifelong learning algorithm to our knowledge.

5 Providing intuition of synergistic learning through simulations

In this section, we explore how relative position of the decision boundaries between two classes in two tasks can affect our proposed approach using simple toy simulations. For simulation study, we have used a deep network (`DN`) architecture with two hidden layers each having 10 nodes.

5.1 Synergistic learning in a simple environment

Consider a very simple two-task environment: Gaussian XOR and Gaussian Exclusive NOR (XNOR) (Figure 4A, see Appendix F for details). The two tasks share the exact same discriminant boundaries: the coordinate axes. Thus, transferring from one task to the other merely requires learning a bit flip of the class labels. We sample a total 750 samples from XOR, followed by another 750 samples from XNOR.

`SynF` and random forests (`RF`) achieve the same generalization error on XOR when training with XOR data (Figure 4Bi). But because `RF` does not account for a change in task, when XNOR data appear, `RF` performance on XOR deteriorates (it catastrophically forgets). In contrast, `SynF` continues to improve on XOR given XNOR data, demonstrating backward transfer. Now consider the generalization error on *XNOR* (Figure 4Bii). Both `SynF` and `RF` are at chance levels for XNOR when only XOR data are available. When XNOR data are available, `RF` must unlearn everything it learned from the XOR data, and thus its performance on XNOR starts out nearly maximally inaccurate, and quickly improves. On the other hand, because `SynF` can leverage the encoder learned using the XOR data, upon getting *any* XNOR data, it immediately performs quite well, and then continues to improve with further XNOR data, demonstrating forward transfer (Figure 4Biii). `SynF` demonstrates positive forward and backward transfer for all sample sizes, whereas `RF` fails to demonstrate forward or backward transfer, and eventually catastrophically forgets the previous tasks. Results for `SynN` and `DN` are qualitatively similar to those of `SynF` and `RF` respectively.

5.2 Synergistic learning in adversarial environments

Statistics has a rich history of *robust learning* (Huber, 1996; Ramoni & Sebastiani, 2001), and machine learning has recently focused on *adversarial learning* (Szegedy et al., 2014; Zhang et al., 2018; 2020; Lowd & Meek, 2005). However, in both cases the focus is on adversarial *examples*, rather than adversarial *tasks*. In the context of synergistic learning, we informally define a task t to be adversarial with respect to task t' if the true joint distribution of task t, without any domain adaptation, impedes performance on task t'. In other words, training data from task t can only add noise, rather than signal, for task t'. An adversarial task for Gaussian XOR is Gaussian XOR rotated by 45° (R-XOR) (Figure 4Aiii). Training on R-XOR therefore impedes the performance of `SynF` and `SynN` on XOR, and thus backward transfer becomes negative, demonstrating

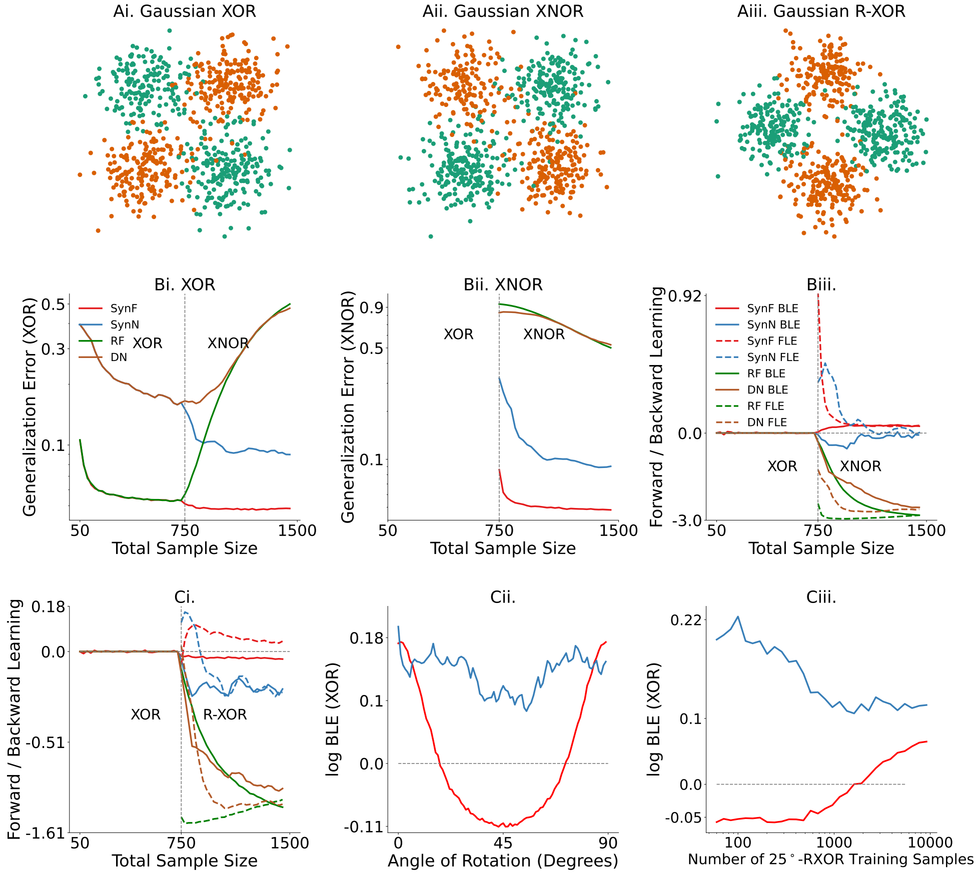

Figure 4: **Synergistic Forest and Synergistic Network demonstrate forward and backward transfer.** The learner is trained from scratch for each sample size so that we can observe the impact of increasing sample size on our algorithms. (*A*) 750 samples from: (*Ai*) Gaussian XOR, (*Aii*) XNOR, which has the same optimal discriminant boundary as XOR, and (*Aiii*) R-XOR, which has a discriminant boundary that is uninformative, and therefore adversarial, to XOR. (*Bi*) Generalization error for XOR, and (*Bii*) XNOR of both `SynF` (red), `RF` (green), `SynN` (blue), `DN` (dark orange). `SynF` outperforms `RF` on XOR when XNOR data is available, and on XNOR when XOR data are available. The same result is true for `SynN` relative to `DN`. (*Biii*) log FLE and log BLE of `SynF` are positive for all sample sizes, and are negative for all sample sizes for `RF`. Again, FLE and BLE is higher for `SynN`compared to those of `DN`. (*Ci*) In an adversarial task setting (100 samples of XOR followed by 100 samples of R-XOR), `SynF`and `SynN` gracefully forgets XOR, whereas `RF`and `DN` demonstrate catastrophic forgetting and interference. (*Cii*) Backward transfer is positive with respect to XOR when the optimal decision boundary of θ-XOR is similar to that of XOR (e.g. angles near 0° and 90°), and negative when the discriminant boundary is uninformative, and therefore adversarial, to XOR (e.g. angles near 45°). (*Ciii*) BLE is a nonlinear function of the source training sample size (XOR sample size is fixed at 500). For `SynN` experiments we did 100 repetitions and reported the results after smoothing it using moving average with a window size of 5. For the `SynF` experiments we used 1000 repetitions and reported the mean of these repetitions.

graceful forgetting (Aljundi et al., 2018) (Figure 4Ci). Because R-XOR is more difficult than XOR for `SynF` (because the discriminant boundaries are oblique (Tomita et al., 2020)), and because the discriminant boundaries are learned imperfectly with finite data, data from XOR can actually improve performance on R-XOR, and thus forward transfer is positive. In contrast, both forward and backward transfer are negative for `RF` and `DN`.

To further investigate this relationship, we design a suite of R-XOR examples, generalizing R-XOR from only 45° to any rotation angle between 0° and 90°, sampling 100 points from XOR, and another 100 from each R-XOR (Figure 4Cii). As the angle increases from 0° to 45°, log BLE flips from positive (≈ 0.18) to negative (≈ -0.11) for SynF. A similar trend is also visible for SynN. The 45°-XOR is the maximally adversarial R-XOR. Thus, as the angle further increases, log BLE increases back up to ≈ 0.18 at 90°, which has an identical discriminant boundary to XOR. Moreover, when θ is fixed at 25°, BLE increases at different rates for different sample sizes of the source task (Figure 4Ciii).

Together, these experiments indicate that the amount of transfer can be a complicated function of (i) the difficulty of learning good representations for each task, (ii) the relationship between the two tasks, and (iii) the sample size of each. Appendix F further investigates this phenomenon in a multi-spiral environment.

6 Benchmark data experiments

For benchmark data, we build SynN encoders using the network architecture described in van de Ven et al. (2020) as “5 convolutional layers followed by two fully-connected layers each containing 2000 nodes with ReLU non-linearities and a softmax output layer”. We use the same network architecture for all the benchmarking models. For the following experiments, we consider two modalities of real data: vision and language. Our language experiments in Appendix G.1 have qualitatively similar results as those of vision experiments, suggesting that SynF and SynN are modality agnostic, sample and computationally efficient lifelong learning algorithms. In addition to the CIFAR 100 dataset, we provide vision experiments on larger datasets with higher sample size per task. However, under the lifelong learning framework, a learning agent, constrained by capacity and computational time, is sequentially trained on multiple tasks. For each task, it has access to limited training samples (Chen & Liu, 2016; Lee et al., 2019; Kemker et al., 2018), and it improves on a particular task by leveraging knowledge from the other tasks. If a learner has enough single task data, it can achieve close to the optimal performance as a single task learner without any doing any sorts of transfer learning and thereby, will not be motivated to look for transfer of knowledge from other task data. Therefore, we are particularly interested in the behavior of our representation ensembling algorithms in the low training sample size regime using CIFAR 100 dataset. The CIFAR 10x10 experiments use only 500 training samples per task. For the corresponding experiments using higher training samples per task (5,000 samples), see Appendix Figure 4. For the FLE curves, we report forward learning efficiency on the corresponding task as that task is introduced. For backward learning efficiency, we evaluate the backward learning efficiency on all of the tasks introduced so far as a new task is introduced. Therefore, for each task the log(BLE) curve starts from 0 when the corresponding task is introduced and goes upward (positive) or downward (negative) as more tasks are seen.

Appendix Table 4, 5, 6 and 7 report the average log learning efficiency after all the tasks have been introduced for different algorithms as a summary.

Table 2: Benchmark dataset details.

Experiment	Dataset	Training samples	Testing samples	Dimension
CIFAR 10X10	CIFAR 100	5000	10000	$3 \times 32 \times 32$
5-dataset	CIFAR-10	50000	10000	$3 \times 32 \times 32$ (resized)
	MNIST	60000	10000	
	SVHN	73257	26032	
	notMNSIT	16853	1873	
	Fashion-MNIST	60000	10000	
Split Mini-Imagenet	Mini-Imagenet	48000	12000	$3 \times 84 \times 84$
FOOD1k 50X20	Food1k	60000	99682	$3 \times 50 \times 50$ (resized)
Spoken Digit	Spoken Digit	1650	1350	28×28 (processed and resized)

6.1 Exploring and explaining transfer capabilities via the CIFAR 10x10 dataset

The CIFAR 100 challenge (Krizhevsky, 2012), consists of 50,000 training and 10,000 test samples, each a 32x32 RGB image of a common object, from one of 100 possible classes, such as apples and bicycles. CIFAR

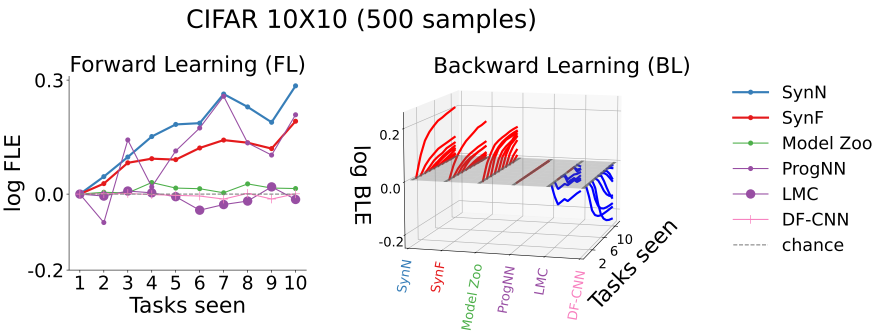

Figure 5: **Performance of different algorithms on the CIFAR 10x10 vision experiments.** *Left*: Forward learning efficiency for various resource building algorithms. *Right*: `SynF` and `SynN` consistently demonstrate backward transfer for each task, whereas other algorithms do not. In all of the plots, the performance of the chance algorithm which chooses a label at random is shown as a horizontal dashed line along 0. The positive and negative values are color coded with red and blue colors, respectively in the right plot.

10x10 divides these data into 10 tasks, each with 10 classes (Lee et al., 2019) (see Appendix G for details). We compare `SynF` and `SynN` to the deep lifelong learning algorithms discussed above.

6.1.1 Resource growing experiments

We first compare `SynF` and `SynN` to several resource growing algorithms: `Model Zoo` (Ramesh & Chaudhari, 2021), `ProgNN` (Rusu et al., 2016), `LMC` (Ostapenko et al., 2021) and `DF-CNN` (Lee et al., 2019) (Figure 5). Both `SynF` and `SynN` demonstrate positive forward transfer for every task (`SynF` increases nearly monotonically), indicating they are robust to distributional shift in ways that `ProgNN` and `DF-CNN` are not. `SynN`, `SynF` and `Model Zoo`(Ramesh & Chaudhari, 2021) demonstrate positive backward transfer, `SynN` is actually monotonically increasing, indicating that with each new task, performance on all prior tasks increases (and `SynF` nearly monotonically increases BLE as well). In contrast, other algorithms, except `Model Zoo`, do not exhibit any positive backward transfer. Overall learning efficiency per task in Figure 1 is the learning efficiency associated with that task having seen all the data. `SynF` and `SynN` both demonstrate positive overall learning efficiency for all tasks (synergistic learning), whereas other algorithms exhibit negative learning efficiency for at least one task (Figure 1 bottom left).

6.1.2 Ablation Experiments

Our proposed algorithms can improve performance on all the tasks (past and future) by both growing additional resources and replaying data from the past tasks. Below we do three ablation experiments using CIFAR 10X10 to measure the relative contribution of resource growth and replay on the performance of our proposed algorithms.

1. **Resource constrained experiments** In this experiment, we devised a "resource constrained" variant of `SynF` experiments to observe the effect of ablating resource growth on `SynF`. In this constrained variant, we compare the lifelong learning algorithm to its single task variant, but ensure that they both have the same amount of resources. For example, on Task 2, we would compare `SynF` with 20 trees (10 trained on 500 samples from Task 1, and another 10 trained on 500 samples from Task 2) to `RF` with 20 trees (all trained on 500 samples Task 2). Although ablating the resource growth results in lower FLE compared to that of its resource growing variant (Figure 5 left and Figure 6 top left), FLE remains positive after enough tasks, and BLE is actually invariant to this change (Figure 6, top left and center). In contrast, all of the reference algorithms that have

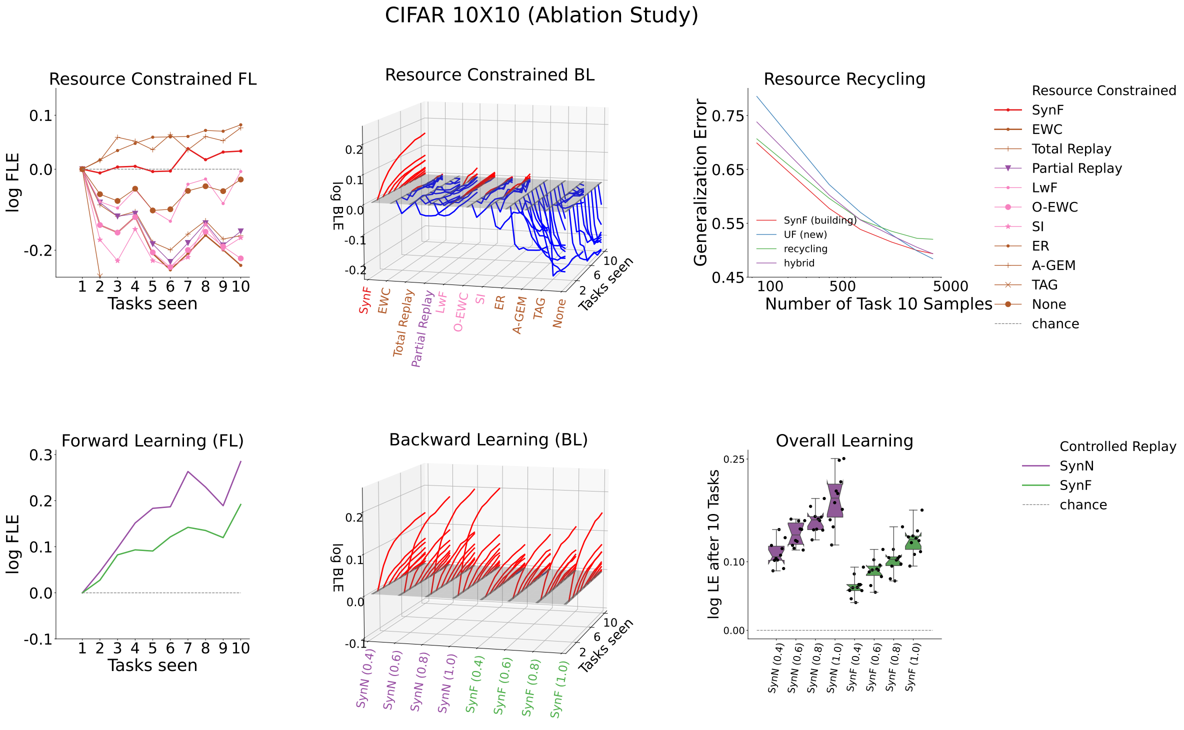

Figure 6: **Ablation experiments on** **SynN and SynF using CIFAR 10X10. Top:** Comparison of Resource Constrained `SynF` with algorithms having a fixed amount of resources. `SynF` is the only approach that demonstrate forward (*Left*) and backward transfer (*Middle*). *Right*: Building and recycling ensembles are two boundaries of a continuum, with hybrid models in the middle. `SynF` achieves lower (better) generalization error than other approaches until 5,000 training samples on the new task are available, but eventually a hybrid approach wins. **Bottom:** Controlled replay experiment on CIFAR 10x10. Fraction of total samples per task replayed is mentioned in parenthesis in the middle and the right plot. *Left*: FLE for each task remains the same for different amount of replay from the old tasks, i.e., the FLE curves for each algorithm with different amount of replay are superimposed on each other. *Middle*: Amount of backward transfer increases as more samples are replayed from the old tasks. The positive and negative values are color coded with red and blue colors, respectively. *Right*: With FLE remaining constant, the overall learning efficiency (LE) increases as the BLEs for different tasks increase with an increasing number of replayed samples.

fixed resources exhibit negative forward and backward transfer. Moreover, the reference algorithms also all exhibit negative final transfer efficiency on each task, whereas our resource constrained `SynF` maintains positive final transfer on every task (Figure 1, bottom left). Interestingly, when using 5,000 samples per task, total and partial replay methods are able to demonstrate positive forward and backward transfer (Supplementary Figure 8), although they require quadratic time. Note that in this experiment, building the single task learners actually requires substantially *more* resources, specifically, $10 + 20 + \cdots + 100 = 550$ trees, as compared with only 100 trees in the prior experiments. In general, to ensure single task learners use the same amount of resources per task as our synergistic learners requires $O(n^2)$ resources, where as `SynF` only requires $O(n)$, a polynomial reduction in resources.

2. **Resource Recycling Experiments** The binary distinction we made above, algorithms either build resources or reallocate them, is a false dichotomy, and biologically unnatural. In biological learning, systems develop from building to fixed resources, as they grow from juveniles to adults. To explore

this continuum of amount of resources to grow, we trained `SynF` on the first nine CIFAR 10x10 tasks using 50 trees per task, with 500 samples per task. For the tenth task, we could (i) select the 50 trees (out of the 450 existing trees) that perform best on task 10 (recycling), (ii) train 50 new trees, as `SynF` would normally do (building), (iii) build 25 and recruit 25 trees (hybrid), or (iv) ignore all prior trees (`RF`). `SynF` outperforms other approaches except when 5,000 training samples are available, but the recycling approach is nearly as good as `SynF` (Figure 6, top right). This result motivates future work to investigate optimal strategies for determining how to optimally leverage existing resources given a new task, and task-unaware settings.

3. **Controlled Replay Experiment** In this experiment, we train 4 different versions of `SynN` and `SynF` sequentially on the 10 tasks from CIFAR 10X10. The only difference between different versions of the algorithms is the amount of old task data replayed. In 4 different versions of each algorithm, we replay 40%, 60%, 80% and 100% of the old task data respectively. As apparent from Figure 6 bottom, replaying old task data has no effect on forward transfer and the proposed algorithms transfer backward more as more data from old tasks are replayed.

6.1.3 Comparison with Single Task Experts

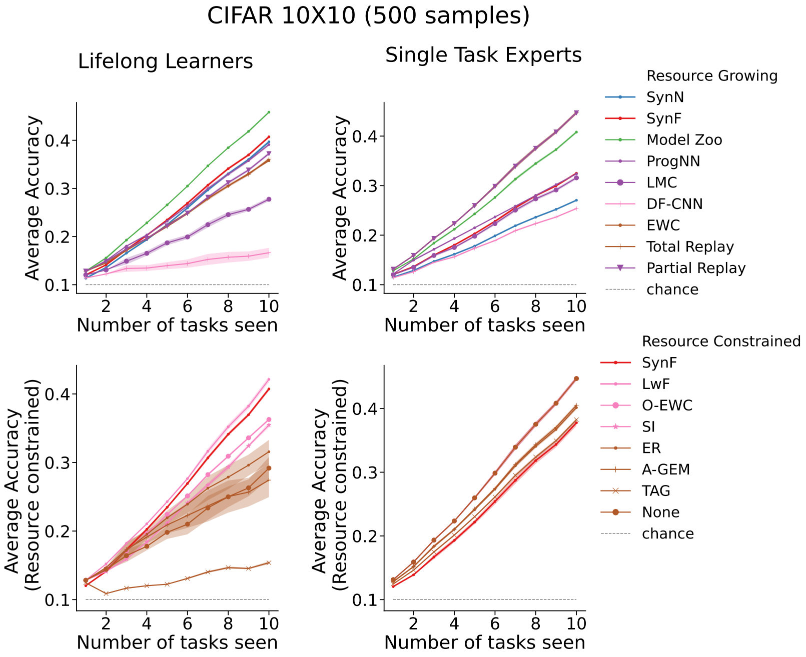

Figure 7: **Average accuracy over 10 tasks as the learners (lifelong and single task experts) see more tasks.** For single task learners, a new stand-alone learner is trained as a new task is seen and the average accuracy over all the task specific learners is reported. `LwF` has the highest multitask accuracy (bottom left) on CIFAR 10X10 while it the best single task accuracy and `SynF` has the lowest single task accuracy (bottom right). Therefore, only accuracy can falsely detect positive transfer. The error bar ($\pm 1.96\times$ std) is shown as a faded color spread centering the mean curve.

In this experiment, we train the lifelong learning algorithms and the single task experts sequentially on 10 tasks from CIFAR 10X10. After each task is introduced, a new single task expert with the same hyperparameters as the corresponding lifelong learner is trained on the task and at the same time, the lifelong learning models are updated for the same task. The lifelong learners and their single task expert counterparts are shown using the same marker and color in Figure 7. Therefore, we will have 10 single task experts for each lifelong learners after 10 tasks are seen. As each new task is seen, we evaluate the performance of the algorithms on 10 tasks and report the average accuracy $\bar{\mathcal{A}}_t$ (Lomonaco & Maltoni, 2017; Maltoni &

Lomonaco, 2019) as:

$$\bar{\mathcal{A}}_t = \frac{1}{T}\sum_{t'=1}^{T}(1 - \mathcal{E}_f^{t'}(\bigcup_{i=1}^{t}\mathbf{S}^i)). \tag{9}$$

If the lifelong learners are able to perform better than their single task expert counterpart, we say the learner successfully transferred knowledge between the tasks. In Figure 7 top left and right, `SynF` `SynN` and `Model Zoo` has better accuracy than their corresponding single task expert. This indicates they can leverage data from other tasks to improve performance in the target task. On the other hand, other algorithms do not improve rather degrade their performance compared to that of the single task experts. Moreover, As apparent from Figure 7 bottom left and right, only multitask accuracy cannot ascertain that one algorithm transfers better than other algorithms. For example, note that in Figure 7 bottom left, `LwF` (Li & Hoiem, 2017) has better average accuracy compared to that of `SynF`. However, as shown in the bottom right of Figure 7, `LwF` has relatively higher single task accuracy compared to that of `SynF`. This is because `LwF` utilizes convolutional layers to extract the local information in the image data while `SynF` does not. Therefore, `LwF` improves accuracy for each task without doing meaningful transfer of information between the tasks. This is evident from the forward and the backward learning efficiency curves in the middle row of Figure 5.

6.1.4 Adversarial experiments

Consider the same CIFAR 10x10 experiments above, but, for Tasks 2 through 9, randomly permute the class labels within each task, rendering each of those tasks adversarial with regard to the first task (because the labels are uninformative). Figure 8A indicates that backward transfer for both `SynF` and `SynN` is invariant to such label shuffling (the other algorithms also seem invariant to label shuffling, but did not demonstrate positive backward transfer). Now, consider a Rotated CIFAR experiment, which uses only data from the first task, divided into two equally sized subsets (making two tasks), where the second subset is rotated by different amounts (Figure 8, right). Learning efficiency of both `SynF` and `SynN` is nearly invariant to rotation angle, whereas the other approaches are far more sensitive to rotation angle. Note that zero rotation angle corresponds to the two tasks *having identical distributions.*

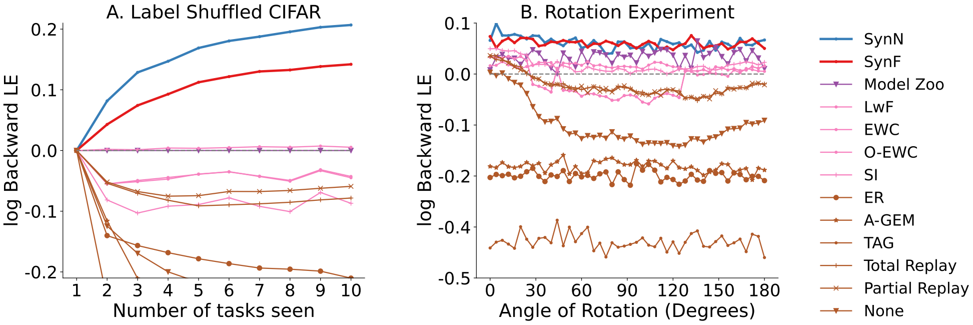

Figure 8: **Extended CIFAR 10x10 experiments.** *A*. Shuffling class labels within tasks two through nine with 500 samples each demonstrates both `SynF` and `SynN` can still achieve positive backward transfer, and that the other algorithms still fail to transfer. *B*. `SynF` and `SynN` are nearly invariant to rotations, whereas other approaches are more sensitive to rotation.

6.2 Further investigating transfer in additional datasets with more dimensions, classes, tasks, and/or samples

6.2.1 5-dataset

In this experiment, we have used **5-dataset** (Malviya et al., 2021) provided in `https://github.com/pranshu28/TAG`. It consists of 5 tasks from five different datasets: CIFAR-10 (Krizhevsky, 2012), MNIST, SVHN (Netzer et al., 2011), notMNIST (Bulatov, 2011), Fashion-MNIST (Xiao et al., 2017). All the monochromatic images were converted to RGB format, and then resized to $3 \times 32 \times 32$. As shown in

Table 2, training samples per task in 5-dataset is relatively higher than that of low data regime typically considered in lifelong learning setting. However, as shown in Figure 9 left column, SynN and SynF show less forgetting than most of the reference algorithms. On the other hand, Model Zoo shows comparatively better performance in relatively high task data size setup. Recall that SynN and SynF are based on bagging, and Model Zoo is based on boosting. With a lower training sample size each member in the ensemble has a high variance which is reduced in the bagging ensemble, and this eventually improves the performance of the bagged ensemble. However, for high sample size, variance for each member is already low, and hence, bagging does not help much. On the other hand, boosting approaches (Model Zoo) iteratively over-fit the learner to the corresponding tasks which results in lower bias. As variance for each member in the ensemble is already low, reducing bias results in better performance for the boosted ensemble. However, lifelong learning models usually have access to a small number of training samples per task as mentioned before in the beginning of Section 6.

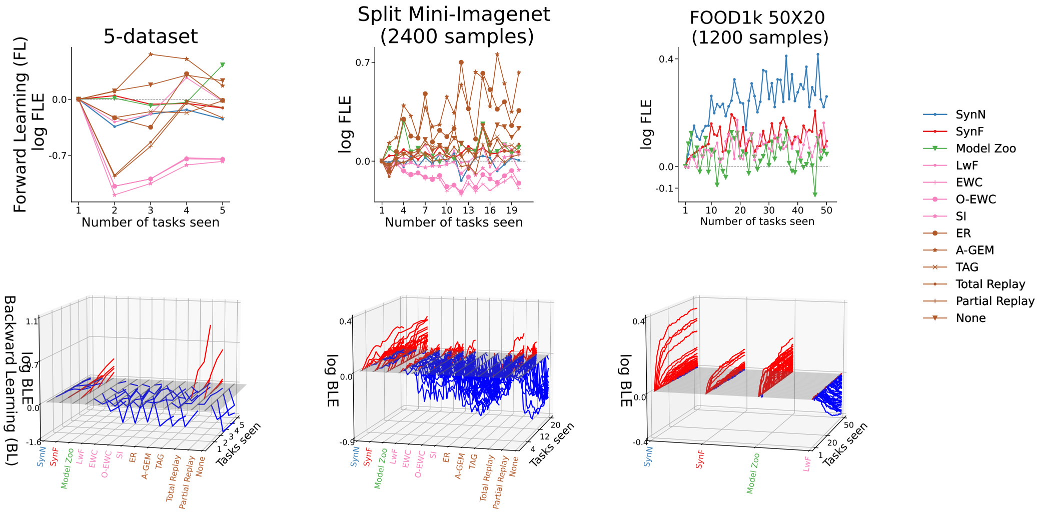

Figure 9: **Performance of different algorithms on the different vision experiments. Right column:** *Top*: A-GEM and None have positive forward transfer for all the tasks while SynN and SynF have gracefully negative forward transfer. *Bottom:* SynF, Model Zoo, Total Replay and Partial Replay show positive backward transfer. The positive and negative values are color coded with red and blue colors, respectively. Sample size for each task is provided in Table 2. **Middle column:** *Top:* A-GEM and ER have better forward transfer compared to that of other algorithms. *Bottom:* SynN, SynF and Model Zoo show positive backward transfer. The positive and negative values are color coded with red and blue colors, respectively. Note that each task in Mini-Imagnet has 2400 training samples which is lower than that of Five Dataset tasks. This relatively lower sample size results in a bit better performance for SynF and SynN compared to those on 5-dataset. **Right column:** *Top:* Both SynN and SynF show positive forward transfer for all the tasks. *Bottom:* SynN enables more backward transfer to the older tasks from the future tasks compared to that of other algorithms. The positive and negative values are color coded with red and blue colors, respectively. SynN performs the best on large scale datasets like food1k.

6.2.2 Split Mini-Imagenet

In this experiment, we have used the **Mini-Imagenet** dataset (Malviya et al., 2021) provided in `https://www.kaggle.com/datasets/whitemoon/miniimagenet`. The dataset was split into 20 tasks each 5 each. Each task has 2400 training samples and 600 testing samples. As shown in Figure 9 middle column, we get positive FLE and BLE for both `SynN` and `SynF`. However, although samples per task is lower compared to that of 5-dataset, it is still quite high. Hence, `Model Zoo`outperforms all the algorithms in this experiment.

6.2.3 FOOD1k 50X20 Dataset

The datasets considered so far are of small scales. In this experiment, we use **Food1k** which is a large scale vision dataset consisting of 1000 food categories from Food2k Min et al. (2021). FOOD1k 50X20 splits these data into 50 tasks with 20 classes each. All the images were resized to the same shape $3 \times 50 \times 50$. For each class, we randomly sampled 60 samples per class for training the models and used rest of the data for testing purpose. Note that so far `Model Zoo` performs the best among the reference resource growing models and `LwF` is the best performing resource constrained algorithm. Therefore, we choose `Model Zoo` and `LwF` as the reference models for the large scale experiment to avoid heavy computational cost. As shown in Figure 9 right column, `SynN` performs the best among all the algorithms. Note that the incremental contributions from additional encoders becomes negligible as `SynN` and `SynF` see a lot of tasks. Hence, the backward learning efficiency for the tasks eventually saturates indicating the possibility of resource recycling stage as mentioned previously in the resource recycling experiments on CIFAR 10X10 vision dataset.

7 Discussion

We introduced quasilinear representation ensembling as an approach to synergistic lifelong learning. Two specific algorithms, `SynF` and `SynN`, achieve both forward and backward transfer, due to leveraging resources (encoders) learned for other tasks without undue computational burdens. In this paper, we have mainly focused on task-aware setting, because it is simpler. Future work will extend our approach to more challenging task-unaware settings. Recycling experiment with CIFAR 10x10 shows that Forest-based representation ensembling approaches can easily add new resources when appropriate. This work therefore motivates additional work on deep learning to enable dynamically adding resources when appropriate and resuse the older representations like the moduler methods (Yoon et al., 2017; Mallya & Lazebnik, 2018; Veniat et al., 2020; Ostapenko et al., 2021).

To achieve backward transfer, `SynF` and `SynN` store old data to update the channel using the newly learned transformers. Because the representation space scales quasilinearly with sample size, storing the data does not increase the space complexity of the algorithm, and it remains quasilinear. It could be argued that by keeping old data and training a model with increasing capacity from scratch (a sequential multitask learning approach), it would be straightforward to maintain performance ($\mathsf{LE} = 1$) in a particular task. However, it is not obvious how to achieve backward transfer with quasilinear time and space complexity even if we are allowed to store all the past data, because computational time would naively become quadratic. The novelty of our proposed approach lies in achieving the aforementioned goal.

For example, both `ProgNN` and Total Replay have quadratic time complexity, unlike `SynF` and `SynN`. Thus, one natural extension of this work would be to incorporate a generative model that could mitigate the need to store all the data.

While we employed quasilinear representation ensembling to address catastrophic forgetting, the paradigm of ensembling *representations* rather than *decisions* can be readily applied more generally. For example, "batch effects" (sources of variability unrelated to the scientific question of interest) have plagued many fields of inquiry, including neuroscience (Bridgeford et al., 2020) and genomics (Johnson et al., 2007). Similarly, federated learning is becoming increasingly central in artificial intelligence, due to its importance in differential privacy (Dwork, 2008). This may be particularly important in light of global pandemics such as COVID-19, where combining small datasets across hospital systems could enable more rapid discoveries (Vogelstein et al., 2020).

Finally, our quasilinear representation ensembling approach closely resembles the constructivist view of brain development (Quartz, 1999; Karmiloff-Smith, 2017). According to this view, the brain goes through progressive elaboration of neural circuits resulting in an augmented cognitive representation while maturing in a certain skill. In a similar way, representation ensembling algorithms can mature in a particular skill such as vision tasks by learning a rich encoder dictionary from different vision datasets and thereby, transfer forward to future or yet unseen vision dataset (see CIFAR 10x10 recruitment experiment as an illustration). However, there is also substantial pruning during development and maturity in the brain circuitry which is important for performance (Sakai, 2020). This motivates future work for pruning encoders to enhance the transferability among tasks even more. Moreover, by carefully designing experiments in which both behaviors and brain are observed while learning across sequences of tasks (possibly in multiple stages of neural development or degeneration), we may be able to learn more about how biological agents are able to transfer forward and backward so efficiently, and transfer that understanding to building more effective artificial intelligences.

References

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (eds.), *Computer Vision – ECCV 2018*, pp. 144–161, Cham, 2018. Springer International Publishing.

Yali Amit and Donald Geman. Shape Quantization and Recognition with Randomized Trees. *Neural Comput.*, 9(7):1545–1588, October 1997.

S. Athey, J. Tibshirani, and S. Wager. Generalized random forests. *Annals of Statistics*, 47(2):1148–1178, 2019.

Diana Benavides-Prado, Yun Sing Koh, and Patricia Riddle. Measuring Cumulative Gain of Knowledgeable Lifelong Learners. In *NeurIPS Continual Learning Workshop*, pp. 1–8, 2018.

Peter J Bickel and Kjell A Doksum. *Mathematical statistics: basic ideas and selected topics, volumes I-II package*. Chapman and Hall/CRC, 2015.

Leo Breiman. Bagging predictors. *Mach. Learn.*, 24(2):123–140, August 1996.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

E W Bridgeford, S Wang, Z Yang, Z Wang, T Xu, and others. Big Data Reproducibility: Applications in Brain Imaging. *bioRxiv*, 2020.

Yaroslav Bulatov. http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html. 2011.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Rich Caruana and Alexandru Niculescu-Mizil. An Empirical Comparison of Supervised Learning Algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pp. 161–168, New York, NY, USA, 2006. ACM.

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.

Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pp. 96–103, New York, New York, USA, July 2008. ACM.

Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, March 2006.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

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*, 2019.

Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, New York, NY, USA, August 2016. Association for Computing Machinery.

Zhiyuan Chen and Bing Liu. Lifelong Machine Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(3):1–145, November 2016. URL `https://doi.org/10.2200/S00737ED1V01Y201610AIM033`.

Kyunghyun Cho, B van Merrienboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

Thomas M Cover and Joy A Thomas. *Elements of Information Theory*. John Wiley & Sons, New York, November 2012.

Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Boosting for transfer learning.(2007), 193–200. In *Proceedings of the 24th international conference on Machine learning*, 2007.

M. Denil, D. Matheson, and N. De Freitas. Narrowing the gap: Random forests in theory and in practice. In Eric P. Xing and Tony Jebara (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 665–673, 6 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL `http://arxiv.org/abs/1810.04805`.

Guneet Singh Dhillon, Pratik Chaudhari, Avinash Ravichandran, and Stefano Soatto. A baseline for few-shot image classification. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rylXBkrYDS`.

Natalia Díaz-Rodríguez, Vincenzo Lomonaco, David Filliat, and Davide Maltoni. Don't forget, there is more than forgetting: new metrics for continual learning. *arXiv preprint arXiv:1810.13166*, 2018.

Cynthia Dwork. Differential Privacy: A Survey of Results. In *Theory and Applications of Models of Computation*, pp. 1–19. Springer Berlin Heidelberg, 2008.

Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems. *J. Mach. Learn. Res.*, 15(1):3133–3181, 2014.

Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1920–1930, Long Beach, California, USA, 06 2019. PMLR. URL `http://proceedings.mlr.press/v97/finn19a.html`.

Y Freund. Boosting a Weak Learning Algorithm by Majority. *Inform. and Comput.*, 121(2):256–285, September 1995.

Peter J Huber. *Robust statistical procedures*, volume 68. Siam, 1996.

W Evan Johnson, Cheng Li, and Ariel Rabinovic. Adjusting batch effects in microarray expression data using empirical Bayes methods. *Biostatistics*, 8(1):118–127, January 2007.

Pearl Judea. What is gained from past learning. *Journal of Causal Inference*, 6(1), 2018.

Annette Karmiloff-Smith. From constructivism to neuroconstructivism: The activity-dependent structuring of the human brain. In *After Piaget*, pp. 1–14. Routledge, 2017.

Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

Way Kuo and Ming J Zuo. *Optimal reliability modeling: principles and applications.* John Wiley & Sons, 2003.

Seungwon Lee, James Stokes, and Eric Eaton. Learning shared knowledge for deep lifelong learning using deconvolutional networks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 2837–2844, 2019.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *Conference on Robot Learning*, pp. 17–26. PMLR, 2017.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *NIPS*, 2017.

Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 641–647, 2005.

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*, pp. 7765–7773, 2018.

Davide Maltoni and Vincenzo Lomonaco. Continuous learning in single-incremental-task scenarios. *Neural Networks*, 116:56–73, 2019.

Pranshu Malviya, Sarath Chandar, and Balaraman Ravindran. Tag: Task-based accumulated gradients for lifelong learning. *arXiv preprint arXiv:2105.05155*, 2021.

Gary Marcus and Ernest Davis. *Rebooting AI: Building Artificial Intelligence We Can Trust.* Pantheon, September 2019.

James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

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.

Nikhil Mehta, Kevin Liang, Vinay Kumar Verma, and Lawrence Carin. Continual learning using a bayesian nonparametric dictionary of weight factors. In *International Conference on Artificial Intelligence and Statistics*, pp. 100–108. PMLR, 2021.

Ronak Mehta, Richard Guo, Cencheng Shen, and Joshua Vogelstein. Estimating information-theoretic quantities with random forests. *arXiv preprint arXiv:1907.00325*, 2019.

Weiqing Min, Zhiling Wang, Yuxin Liu, Mengjiang Luo, Liping Kang, Xiaoming Wei, Xiaolin Wei, and Shuqiang Jiang. Large scale visual food recognition. *CoRR*, abs/2103.16107, 2021.

Tom M Mitchell. Machine learning and data mining. *Communications of the ACM*, 42(11):30–36, 1999.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, November 2018. URL `https://market.android.com/details?id=book-dWB9DwAAQBAJ`.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Oleksiy Ostapenko, Pau Rodriguez, Massimo Caccia, and Laurent Charlin. Continual learning via local module composition. *Advances in Neural Information Processing Systems*, 34:30298–30312, 2021.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 2019.

Judea Pearl. The seven tools of causal inference, with reflections on machine learning. *Commun. ACM*, February 2019.

Robi Polikar, Lalita Upda, Satish S Upda, and Vasant Honavar. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 31(4):497–508, 2001.

Cristhian Potes, Saman Parvaneh, Asif Rahman, and Bryan Conroy. Ensemble of feature-based and deep learning-based classifiers for detection of abnormal heart sounds. In *2016 computing in cardiology conference (CinC)*, pp. 621–624. IEEE, 2016.

Carey E Priebe, Joshua T Vogelstein, Florian Engert, and Christopher M White. Modern Machine Learning: Partition & Vote. September 2020.

Xueheng Qiu, Le Zhang, Ye Ren, Ponnuthurai N Suganthan, and Gehan Amaratunga. Ensemble deep learning for regression and time series forecasting. In *2014 IEEE symposium on computational intelligence in ensemble learning (CIEL)*, pp. 1–6. IEEE, 2014.

Steven R Quartz. The constructivist brain. *Trends in cognitive sciences*, 3(2):48–57, 1999.

Rahul Ramesh and Pratik Chaudhari. Model zoo: A growing brain that learns continually. In *International Conference on Learning Representations*, 2021.

Marco Ramoni and Paola Sebastiani. Robust learning with missing data. *Machine Learning*, 45(2):147–170, 2001.

Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Paul Ruvolo and Eric Eaton. ELLA: An Efficient Lifelong Learning Algorithm. In *International Conference on Machine Learning*, volume 28, pp. 507–515, February 2013. URL `http://proceedings.mlr.press/v28/ruvolo13.html`.

Jill Sakai. Core Concept: How synaptic pruning shapes neural wiring during development and, possibly, in disease. *Proc. Natl. Acad. Sci. U. S. A.*, 117(28):16096–16099, July 2020.

Jonathan Schwarz, Jelena Luketina, Wojciech M Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. *arXiv preprint arXiv:1805.06370*, 2018.

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.

Shagun Sodhani, Sarath Chandar, and Yoshua Bengio. Toward training recurrent neural networks for lifelong learning. *Neural computation*, 32(1):1–35, 2020.

Charles J Stone. Consistent Nonparametric Regression. *Ann. Stat.*, 5(4):595–620, July 1977.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*, 01 2014.

Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in neural information processing systems*, pp. 640–646, 1996.

Sebastian Thrun and Lorien Pratt. *Learning to Learn*. Springer Science & Business Media, December 2012. URL `https://market.android.com/details?id=book-X_jpBwAAQBAJ`.

Tyler M Tomita, James Browne, Cencheng Shen, Jaewon Chung, Jesse L Patsolic, Benjamin Falk, Jason Yim, Carey E Priebe, Randal Burns, Mauro Maggioni, and Joshua T Vogelstein. Sparse Projection Oblique Randomer Forests. *J. Mach. Learn. Res.*, 2020.

L G Valiant. A Theory of the Learnable. *Commun. ACM*, 27(11):1134–1142, November 1984. URL `http://doi.acm.org/10.1145/1968.1972`.

Gido M. van de Ven and Andreas S. Tolias. Three scenarios for continual learning. *CoRR*, abs/1904.07734, 2019. URL `http://arxiv.org/abs/1904.07734`.

Gido M van de Ven, Hava T Siegelmann, and Andreas S Tolias. Brain-inspired replay for continual learning with artificial neural networks. *Nature communications*, 11:4069, 2020.

Gido M van de Ven, Tinne Tuytelaars, and Andreas S Tolias. Three types of incremental learning. *Nature Machine Intelligence*, pp. 1–13, 2022.

Iris van Rooij, Mark Blokpoel, Johan Kwisthout, and Todd Wareham. *Cognition and Intractability: A Guide to Classical and Parameterized Complexity Analysis*. Cambridge University Press, April 2019.

V Vapnik and A Chervonenkis. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory Probab. Appl.*, 16(2):264–280, January 1971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł Ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017.

Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors. *arXiv preprint arXiv:2012.12631*, 2020.

Joshua T Vogelstein, Michael Powell, Allison Koenecke, Ruoxuan Xiong, Nicole Fischer, Sakibul Huq, Adham M Khalafallah, Brian Caffo, Elizabeth A Stuart, Nickolas Papadopoulos, Kenneth W Kinzler, Bert Vogelstein, Shibin Zhou, Chetan Bettegowda, Maximilian F Konig, Brett Mensh, and Susan Athey. Alpha-1 adrenergic receptor antagonists for preventing acute respiratory distress syndrome and death from cytokine storm syndrome. *ArXiv*, April 2020.

Haixun Wang, Wei Fan, Philip S Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 226–235, 2003.

Abraham J Wyner, Matthew Olson, Justin Bleich, and David Mease. Explaining the success of adaboost and random forests as interpolating classifiers. *The Journal of Machine Learning Research*, 18(1):1558–1590, 2017.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3014–3023, 2021.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong Learning with Dynamically Expandable Networks. *International Conference on Learning Representations*, August 2017.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3987–3995. JMLR. org, 2017.

Chen Zeno, Itay Golan, Elad Hoffer, and Daniel Soudry. Task Agnostic Continual Learning Using Online Variational Bayes. *arXiv*, March 2018.

Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. Mitigating unwanted biases with adversarial learning. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 335–340, 2018.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

Jingfeng Zhang, Xilie Xu, Bo Han, Gang Niu, Lizhen Cui, Masashi Sugiyama, and Mohan Kankanhalli. Attacks which do not kill training make adversarial learning stronger. In *International conference on machine learning*, pp. 11278–11287. PMLR, 2020.

Jing Zhao, Blanca Quiroz, L Quentin Dixon, and R Malatesha Joshi. Comparing Bilingual to Monolingual Learners on English Spelling: A Meta-analytic Review. *Dyslexia*, 22(3):193–213, August 2016.

A Decision Tree as a Compositional Hypothesis

Consider learning a decision tree for a two class classification problem. The input to the decision tree is a set of n feature-vector/response pairs, (x_i, y_i). The learned tree structure corresponds to the encoder u, because the tree structure maps each input feature vector into an indicator encoding in which leaf node each feature vector resides. Formally, $u : \mathcal{X} \mapsto [L]$, where $[L] = \{\mathbb{1}_{\{\mathcal{X}\in l_1\}}, \mathbb{1}_{\{\mathcal{X}\in l_2\}}, \ldots, \mathbb{1}_{\{\mathcal{X}\in l_L\}}\}$ and L is the total number of leaf nodes. In other words, u maps from the original data space, to a L-dimensional one-hot encoded sparse binary vector, where the sole non-zero entry indicates in which leaf node a particular observation falls, that is, $\tilde{x} := u(x) \in \{0,1\}^L$ where $\|\tilde{x}\| = 1$.

Learning the channel is simply a matter of counting the fraction of observations in each leaf per class. So, the channel is trained using n pairs of transformed feature-vector/response pairs $(\tilde{x}_i, y_i)$, and it assigns a probability of each class in each leaf: $v_l := \mathbb{P}[y_i = 1|\tilde{x}_i = l], \forall l \in \{1, 2, \cdots, L\}$ and $v(\tilde{x}) = \bigcup_{l=1}^{L} v_l$. In other words, for two class classification, v maps from the L-dimensional binary vector to the probability that x is in class 1. The decider is simply $w\left(v(\tilde{x})\right) = \mathbb{1}_{\{v(\tilde{x})>0.5\}}$, that is, it outputs the most likely class label of the leaf node that x falls into.

For inference, the tree is given a single x, and it is passed down the tree until it reaches a leaf node, where it is represented by its leaf identifier $\tilde{x}$. The channel takes $\tilde{x}$ as input, and outputs the estimated posterior probability of being in class 1 for the leaf node in which $\tilde{x}$ resides: $v(\tilde{x}) = \mathbb{P}[y = 1|\tilde{x}]$. If $v(\tilde{x})$ is bigger than 0.5, the decider decides that x is in class 1, and otherwise, it decides it is in class 0.

B Compositional Representation Ensembling

Consider a scenario in which we have two tasks, one following the other. Assume that we already learned a single decomposable hypothesis for the first task: $w_1 \circ v_1 \circ u_1$, and then we get new data associated with a

Algorithm 1 Add a new SynX encoder for a task. OOB = out-of-bag.

```
Require:
    (1) $t$                                                        ▷ current task number
    (2) $\mathcal{D}_n^t = (\mathbf{x}^t, \mathbf{y}^t) \in \mathbb{R}^{n \times p} \times \{1, \ldots, K\}^n$    ▷ training data for task $t$
Ensure:
    (1) $u_t$                                                      ▷ an encoder trained on task $t$
    (2) $\mathcal{I}_{OOB}^t$                                      ▷ a set of the indices of OOB data
function SynX.fit($t, (\mathbf{x}^t, \mathbf{y}^t)$)
    $u_t, \mathcal{I}_{OOB}^t \leftarrow$ encoder.fit($\mathbf{x}^t$, $\mathbf{y}^t$)   ▷ train an encoder on partitioned data
    return $u_t, \mathcal{I}_{OOB}^t$
end function
```

second task. Let n_1 denote the sample size for the first task, and n_2 denote the sample size for the second task, and $n = n_1 + n_2$. The representation ensembling approach generally works as follows. First, since we want to transfer forward to the second task, we push all the new data through the first encoder u_1, which yields $\tilde{x}_{n_1+1}^{(1)}, \ldots, \tilde{x}_n^{(1)}$. Second, we learn a new encoder u_2 using the new data, $\{(x_i, y_i)\}_{i=n_1+1}^n$. We then push the new data through the new encoder, yielding $\tilde{x}_{n_1+1}^{(2)}, \ldots, \tilde{x}_n^{(2)}$. Third, we train a new channel, v_2. To do so, v_2 is trained on the outputs from both encoders, that is, $\{(\tilde{x}_i^{(j)}, y_i)\}_{i=n_1+1}^n$ for $j = 1, 2$. The output of v_2 for any new input x is the posterior probability (or score) for that point for each potential response in task two (class label). Thus, by virtue of ensembling these representations, this approach enables forward transfer (Rusu et al., 2016; Dhillon et al., 2020).

Now, we would also like to improve performance on the first task using the second task's data. While many lifelong methods have tried to achieve this kind of backward transfer, to date, they have mostly failed (Ruvolo & Eaton, 2013). Recall that previously we had already pushed all the first task data through the first task encoder, which had yielded $\tilde{x}_1^{(1)}, \ldots, \tilde{x}_{n_1}^{(1)}$. Assuming we kept any of the first task's data, or can adequately simulate it, we can push those data through u_2 to get a second representation of the first task's data: $\tilde{x}_1^{(2)}, \ldots, \tilde{x}_{n_1}^{(2)}$. Then, v_1 would be trained on both representations of the first task's data. This 'replay-like' procedure facilitates backward transfer, that is, improving performance on previous tasks by leveraging data from newer tasks. Both the forward and backward transfer updates can be implemented every time we obtain data associated with a new task. Enabling the channels to ensemble *omnidirectionally* between all sets of tasks is the key innovation of our proposed synergistic learning approaches.

C Synergistic Algorithms

In this paper, we have proposed two concrete synergistic algorithms, Synergistic Forests (SynF) and Synergistic Networks (SynN). The two algorithms differ in their details of how to update encoders and channels, but abstracting a level up they are both special cases of the same procedure. Let SynX refer to any possible synergistic algorithm. Algorithms 1, 2, 3, and 4 provide pseudocode for adding encoders, updating channels, and making predictions for any SynX algorithm. Whenever the learner gets access to a new task data, we use Algorithm 1 to train a new encoder for the corresponding task. We split the data into two portion– one set is used to learn the encoder and the other portion is called the held out or out-of-bag (OOB) data which is returned by Algorithm 1 to be used by Algorithm 3 to learn the channel for the corresponding task. Note that we push the OOB data through the in-task encoder and the whole dataset through the cross-task encoders to update the channel, i.e, learn the posteriors according to the new encoder. Then we use Algorithm 3 to replay the old task data through the new encoder and update their corresponding channels. Finally, while predicting for a test sample, we use Algorithm 4. Given the task identity, we use the corresponding channel to get the average estimated posterior and predict the class label as the arg max of the estimated posteriors.

Algorithm 2 Add a new `SynX` channel for the current task.

```
Require:
    (1) $t$                                                        ▷ current task number
    (2) $\boldsymbol{u}_t = \{u_t\}_{t'=1}^{t}$                    ▷ the set of encoders
    (3) $\mathcal{D}_n^t = (\mathbf{x}^t, \mathbf{y}^t) \in \mathbb{R}^{n \times p} \times \{1, \ldots, K\}^n$   ▷ training data for task $t$
    (4) $\mathcal{I}_{OOB}^t$                                      ▷ a set of the indices of OOB data for the current task
Ensure: $\boldsymbol{v}_t = \{v_{t,t'}\}_{t'=1}^{t}$   ▷ in-task ($t' = t$) and cross-task ($t' \neq t$) channels for task $t$
function SynX.ADD__CHANNEL($t, \boldsymbol{u}_t, (\mathbf{x}_t, \mathbf{y}_t), \mathcal{I}_{OOB}^t$)
    $v_{tt} \leftarrow u_t$.add_channel(($\mathbf{x}_t, \mathbf{y}_t$), $\mathcal{I}_{OOB}^t$)   ▷ add the in-task channel using OOB data
    for $t' = 1, \ldots, t-1$ do                                   ▷ update the cross task channels for task $t$
        $v_{tt'} \leftarrow u_{t'}$.add_channel($\mathbf{x}_t, \mathbf{y}_t$)
    end for
    return $v_t$
end function
```

Algorithm 3 Update `SynX` channel for the previous tasks.

```
Require:
    (1) $t$                                                        ▷ current task number
    (2) $u_t$                                                      ▷ encoder for the current task
    (3) $\mathcal{D} = \{\mathcal{D}^{t'}\}_{t'=1}^{t-1}$          ▷ training data for tasks $t' = 1, \cdots, t-1$
Ensure: $\boldsymbol{v} = \{\boldsymbol{v}_{t'}\}_{t'=1}^{t-1}$   ▷ all previous task voters
function SynX.UPDATE__CHANNEL($t, u_t, \mathcal{D}$)
    for $t' = 1, \ldots, t-1$ do                                   ▷ update the cross task channels
        $v_{t't} \leftarrow u_t$.get_channel($\mathbf{x}_{t'}, \mathbf{y}_{t'}$)
    end for
    return $\boldsymbol{v}$
end function
```

D Reference Algorithm Implementation Details

The same network architecture was used for all compared deep learning methods. Following van de Ven et al. (2020), the 'base network architecture' consisted of five convolutional layers followed by two-fully connected layers each containing 2000 nodes with ReLU non-linearities and a softmax output layer. The convolutional layers had 16, 32, 64, 128 and 254 channels, they used batch-norm and a ReLU non-linearity, they had a 3x3 kernel, a padding of 1 and a stride of 2 (except the first layer, which had a stride of 1). This architecture was used with a multi-headed output layer (i.e., a different output layer for each task) for all algorithms using a fixed-size network. For ProgNN and DF-CNN the same architecture was used for each column introduced for each new task, and in our `SynN` this architecture was used for the transformers u_t (see above). In these implementations, ProgNN and DF-CNN have the same architecture for each column introduced for each task. Among the reference algorithms, `EWC`, `O-EWC`, `LwF`, `SI`, `Total Replay` and `Partial Replay` results were produced using the repository `https://github.com/GMvandeVen/progressive-learning-pytorch`. For `ProgNN` and `DF-CNN` we used the code provided in `https://github.com/Lifelong-ML/DF-CNN`. For all other reference algorithms, we modified the code provided by the authors to match the deep net architecture as mentioned above and used the default hyperparameters provided in the code.

E Training Time Complexity Analysis

Consider a lifelong learning environment with T tasks each with n' samples, i.e., total training samples, $n = n'T$. For all the algorithm with time complexity $\tilde{\mathcal{O}}(n)$, the training time grows linearly with more training samples. We discuss all other algorithms with non-linear time complexity below.

Algorithm 4 Predicting a class label using `SynX`.

```
Require:
    (1) $x \in \mathbb{R}^p$                      ▷ test datum
    (2) $t$                                       ▷ task identity associated with $x$
    (3) $\boldsymbol{u}$                          ▷ all $T$ reperesenters
    (4) $\boldsymbol{v}_t$                        ▷ channel for task $t$
Ensure: $\hat{y}$                                 ▷ a predicted class label
function $\hat{y}$ = SynX.predict($t, x, v_t$)
    $T \leftarrow$ SynX.get_task_number()         ▷ get the total number of tasks
    $\hat{\mathbf{p}}_t = \mathbf{0}$             ▷ $\hat{\mathbf{p}}_t$ is a $K$-dimensional posterior vector
    for $t' = 1, \ldots, T$ do                    ▷ aggregate the posteriors calculated from $T$-th task channel
        $\hat{\mathbf{p}}_t \leftarrow \hat{\mathbf{p}}_t + v_{tt'}$.predict_proba($u_{t'}(x)$)
    end for
    $\hat{\mathbf{p}}_t \leftarrow \hat{\mathbf{p}}_t / T$
    $\hat{y} = \arg\max_i(\hat{\mathbf{p}}_t)$    ▷ find the index $i$ of the elements in the vector $\hat{\mathbf{p}}_t$ with maximum probability
    return $\hat{y}$
end function
```

Table 1: Hyperparameters for `SynF` in CIFAR-10X10 experiments. n_estimators is denoted by B, the number of trees, above.

Hyperparameters	**Value**
n_estimators (500 training samples per task)	10
n_estimators (5000 training samples per task)	40
max_depth	30
max_samples (OOB split)	0.67
min_samples_leaf	1

E.1 EWC

Consider the time required to train the weights for each task in `EWC` is $k_c n'$ and each task adds additional $k_l n'$ time from the regularization term. Here, k_c and k_l are both constants. Therefore, time required to learn all the T tasks can be written as:

$$\begin{aligned}
& k_c n' + (k_c n' + k_l n') + \cdots + (k_c n' + (T-1) k_l n') \\
& = k_c n' T + k_l n' \sum_{t=1}^{T-1} t \\
& = k_c n' T + k_l n' \frac{T(T-1)}{2} \\
& = k_c n + 0.5 k_l n T - 0.5 k_l n \\
& = \tilde{\mathcal{O}}(nT). \qquad (10)
\end{aligned}$$

E.2 Total Replay

Consider the time to train the model on n' samples is $k_c n'$. Therefore, time required to learn all the T tasks can be written as:

Table 2: Hyperparameters for **SynF** in Five Datasets, Split Mini-Imagenet, FOOD1k experiments. n_estimators is denoted by B, the number of trees, above. Note that we use the same hyperparameters for all of the aforementioned datasets.

Hyperparameters	**Value**
n_estimators	10
max_depth	30
max_samples (OOB split)	0.67
min_samples_leaf	1

Table 3: Hyperparameters for `SynN` in CIFAR 10X10, Five Datasets, Split Mini-Imagenet, FOOD1k experiments. Note that we use the same hyperparameters for all of the aforementioned datasets.

Hyperparameters	**Value**
optimizer	Adam
learning rate	3×10^{-4}
max_samples (OOB split)	0.67
K (KNN channel)	$\log_2$(number of samples per task)

$$\begin{aligned}
&k_c n' + k_c(n' + n') + \cdots + k_c n' T \\
&= k_c n' \sum_{t=1}^{T} t \\
&= k_c n' \frac{T(T+1)}{2} \\
&= 0.5 k_c n T + 0.5 k_c n \\
&= \tilde{\mathcal{O}}(nT) \qquad (11)
\end{aligned}$$

E.3 `ProgNN`

Consider the time required to train each column in `ProgNN` is $k_c n'$ and each lateral connection can be learned with time $k_l n'$. Therefore, time required to learn all the T tasks can be written as:

$$\begin{aligned}
&k_c n' + (k_c n' + k_l n') + \cdots + (k_c n' + (T-1) k_l n') \\
&= k_c n' T + k_l n' \sum_{t=1}^{T-1} t \\
&= k_c n' T + k_l n' \frac{T(T-1)}{2} \\
&= k_c n + 0.5 k_l n T - 0.5 k_l n \\
&= \tilde{\mathcal{O}}(nT) \qquad (12)
\end{aligned}$$

F Simulated Results

In each simulation, we constructed an environment with two tasks. For each, we sample 750 times from the first task, followed by 750 times from the second task. These 1,500 samples comprise the training data. We sample another 1,000 hold out samples to evaluate the algorithms. We fit a random forest (`RF`) (technically, an uncertainty forest which is an honest forest with a finite-sample correction (Mehta et al., 2019)) and a SynF. We repeat this process 30 times to obtain errorbars. Error bars in all cases were negligible.

F.1 Gaussian XOR

Gaussian XOR is two class classification problem with equal class priors. Conditioned on being in class 0, a sample is drawn from a mixture of two Gaussians with means $\pm \begin{bmatrix}0.5, & 0.5\end{bmatrix}^\mathsf{T}$, and variances proportional to the identity matrix. Conditioned on being in class 1, a sample is drawn from a mixture of two Gaussians with means $\pm \begin{bmatrix}0.5, & -0.5\end{bmatrix}^\mathsf{T}$, and variances proportional to the identity matrix. Gaussian XNOR is the same distribution as Gaussian XOR with the class labels flipped. Rotated XOR (R-XOR) rotates XOR by θ° degrees.

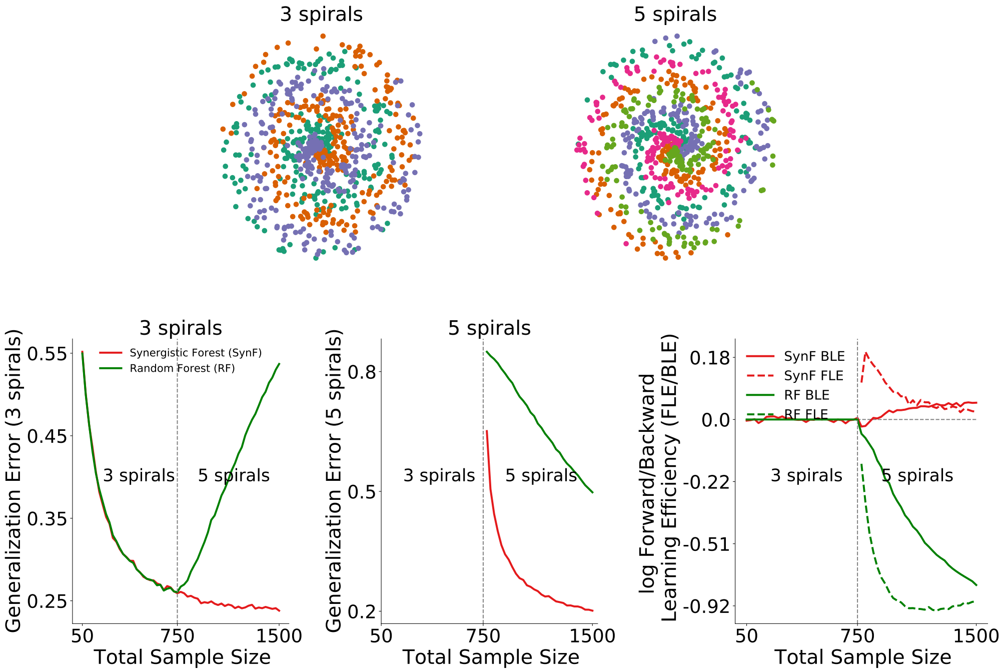

Figure 1: *Top*: 750 samples from 3 spirals (left) and 5 spirals (right). *Bottom left*: SynF outperforms RF on 3 spirals when 5 spirals data is available, demonstrating *backward* transfer in SynF. *Bottom center*: SynF outperforms RF on 5 spirals when 3 spirals data is available, demonstrating *forward* transfer in SynF. *Bottom right*: Transfer Efficiency of SynF. The forward (solid) and backward (dashed) curves are the ratio of the generalization error of SynF to RF in their respective figures. SynF demonstrates decreasing forward transfer and increasing backward transfer in this environment.

F.2 Spirals

A description of the distributions for the two tasks is as follows: let K be the number of classes and $S \sim$ multinomial$(\frac{1}{K}\vec{1}_K, n)$. Conditioned on S, each feature vector is parameterized by two variables, the radius r and an angle θ. For each sample, r is sampled uniformly in $[0, 1]$. Conditioned on a particular class, the angles are evenly spaced between $\frac{4\pi(k-1)t_K}{K}$ and $\frac{4\pi(k)t_K}{K}$ where t_K controls the number of turns in the spiral. To inject noise along the spiral, we add Gaussian noise to the evenly spaced angles $\theta' : \theta = \theta' + \mathcal{N}(0, \sigma_K^2)$. The observed feature vector is then $(r \ \cos(\theta), r \ \sin(\theta))$. In Figure 1 we set $t_3 = 2.5$, $t_5 = 3.5$, $\sigma_3^2 = 3$ and $\sigma_5^2 = 1.876$.

Consider an environment with a three spiral and five spiral task (Figure 1). In this environment, axis-aligned splits are inefficient, because the optimal partitions are better approximated by irregular polytopes than by the orthotopes provided by axis-aligned splits. The three spiral data helps the five spiral performance because the optimal partitioning for these two tasks is relatively similar to one another, as indicated by positive forward transfer. This is despite the fact that the five spiral task requires more fine partitioning than the three spiral task. Because SynF grows relatively deep trees, it over-partitions space, thereby rendering tasks with more coarse optimal decision boundaries useful for tasks with more fine optimal decision boundaries. The five spiral data also improves the three spiral performance.

G Real Data Extended Results

Table 4: **Performance metric: average** log(LE) **after** 10 **tasks calculated for different algorithms on CIFAR 10x10 (**500 **samples per task).**

Algorithms	log(LE)(±std)
SynN	**0.19**(±0.04)
SynF	**0.13**(±0.02)
Model Zoo	0.09(±0.04)
ProgNN	0.11(±0.09)
LMC	−0.05(±0.04)
DF-CNN	−0.11(±0.08)
EWC	−0.15(±0.04)
Total Replay	−0.15(±0.03)
Partial Replay	−0.13(±0.03)
SynF(resource constrained)	**0.05**(±0.03)
LwF	−0.05(±0.03)
O-EWC	−0.14(±0.04)
SI	−0.16(±0.03)
ER	−0.13(±0.12)
A-GEM	−0.19(±0.14)
TAG	−0.32(±0.04)
None	−0.24(±0.10)

Table 5: **Performance metric: average** log(LE) **after** 10 **tasks calculated for different algorithms on 5-dataset.**

Algorithms	log(LE)(±std)
SynN	−**0.27**(±0.22)
SynF	−**0.05**(±0.11)
Model Zoo	0.24(±0.12)
EWC	−1.06(±0.60)
Total Replay	−0.18(±0.22)
Partial Replay	−0.27(±0.26)
LwF	−0.39(±0.48)
O-EWC	−1.07(±0.60)
SI	−1.15(±0.69)
ER	−0.78(±1.03)
A-GEM	−0.55(±0.90)
TAG	−0.56(±0.58)
None	−1.15(±1.30)

G.1 Spoken Digit experiment

In this experiment, we used the **Spoken Digit** dataset provided in `https://github.com/Jakobovski/free-spoken-digit-dataset`. The dataset contains audio recordings from 6 different speakers with 50 recordings for each digit per speaker (3000 recordings in total). The experiment was set up with 6 tasks

Table 6: **Performance metric: average** log(LE) **after** 10 **tasks calculated for different algorithms on Split Mini-Imagenet.**

Algorithms	log(LE)(±std)
SynN	**0.02**(±0.10)
SynF	**0.10**(±0.04)
Model Zoo	0.23(±0.10)
EWC	−0.29(±0.12)
Total Replay	0.06(±0.13)
Partial Replay	0.00(±0.10)
LwF	0.02(±0.08)
O-EWC	−0.21(±0.10)
SI	−0.14(±0.12)
ER	−0.02(±0.27)
A-GEM	0.06(±0.26)
TAG	−0.05(±0.15)
None	−0.22(±0.23)

Table 7: **Performance metric: average** log(LE) **after** 10 **tasks calculated for different algorithms on FOOD1k 50X20.**

Algorithms	log(LE)(±std)
SynN	**0.31**(±0.06)
SynF	**0.14**(±0.04)
Model Zoo	0.13(±0.08)
LwF	−0.06(±0.13)

where each task contains recordings from only one speaker. For each recording, a spectrogram was extracted using Hanning windows of duration 16 ms with an overlap of 4 ms between the adjacent windows. The spectrograms were resized down to 28×28. The extracted spectrograms from 8 random recordings of '5' for 6 speakers are shown in Figure 2. For each Monte Carlo repetition of the experiment, spectrograms extracted for each task were randomly divided into 55% train and 45% test set. We have provided benchmarking for seven algorithms out of the 11 algorithms as mentioned in Subsection 2.3. As shown in Figure 3 and main text Figure 2, both SynF and SynN show positive transfer and synergistic learning between the spoken digit tasks, in contrast to other methods, some of which show only forward transfer, others show only backward transfer, with none showing both, and some showing neither.

G.2 CIFAR 10x10

Supplementary Table 9 shows the image classes associated with each task number. Supplementary Figure 4 is the same as Figure 5 but with 5,000 training samples per task, rather than 500. Notably, with 5,000 samples, replay methods and Model Zoo are able to transfer both forward and backward as well. However, note that although total replay outperforms both SynF and SynN with large sample sizes, it is not a *bona fide* lifelong learning algorithm, because it requires n^2 time. Moreover, the replay methods will eventually forget as more tasks are introduced because it will run out of capacity.

G.3 CIFAR Label Shuffling

Supplementary Figure 5 shows the same result as the label shuffling from Figure 8, but with 5,000 samples per class. The results for SynN and SynF are qualitatively similar, in that they transfer backward. The replay methods are also able to transfer when using this larger number of samples, although with considerably higher computational cost.

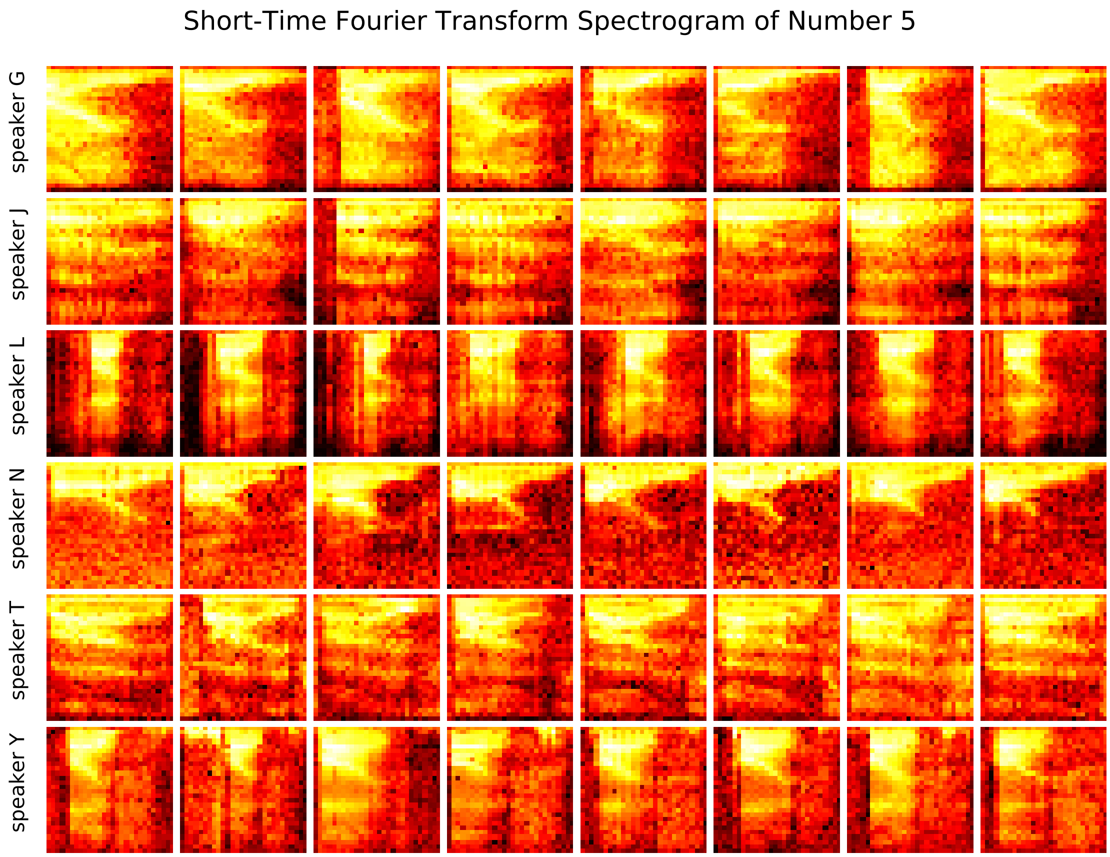

Figure 2: Spectrogram extracted from 8 different recordings of 6 speakers uttering the digit '5'.

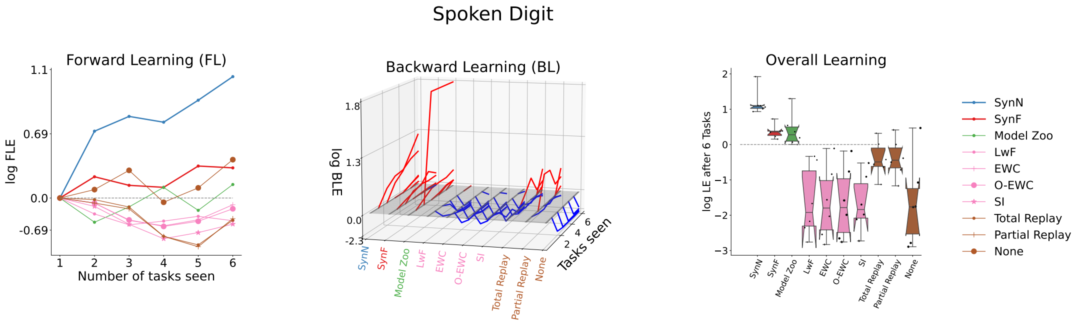

Figure 3: **Performance of different algorithms on the Spoken Digit experiments.** The positive and negative values are color coded with red and blue colors, respectively in the top middle panels. Both SYNF and SYNN show positive forward and backward transfer as well as synergistic learning for the spoken digit tasks, in contrast to other seven methods, some of which show only forward transfer, others show only backward transfer, with none showing both, and some showing neither.

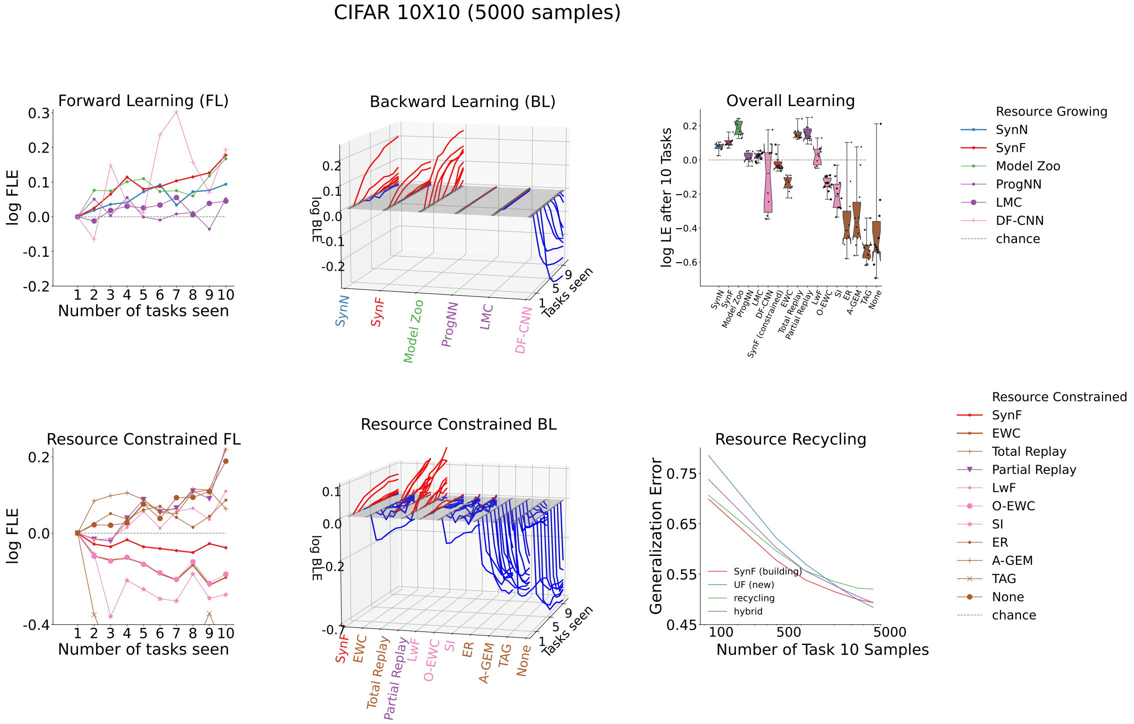

Figure 4: Performance of different algorithms on CIFAR 10x10 vision dataset for 5,000 training samples per task. SynN maintains approximately the same forward transfer (top left and middle left) and backward transfer (top and middle row second column) efficiency as those for 500 samples per task whereas other algorithms show reduced or nearly unchanged transfer. SynF still demonstrates positive forward, backward, and final transfer, unlike most of the state-of-the-art algorithms, which demonstrate forgetting. The replay methods, however, do demonstrate transfer, albeit with significantly higher computational cost.

Table 8: Hyperparameters for SynF in spoken digit experiment.

Hyperparameters	**Value**
n_estimators (275 training samples per task)	10
max_depth	30
max_samples (OOB split)	0.67
min_samples_leaf	1

Table 9: Task splits for CIFAR 10x10.

Task #	Image Classes
1	apple, aquarium fish, baby, bear, beaver, bed, bee, beetle, bicycle, bottle
2	bowl, boy, bridge, bus, butterfly, camel, can, castle, caterpillar
3	chair, chimpanzee, clock, cloud, cockroach, couch, crab, crocodile, cup, dinosaur
4	dolphin, elephant, flatfish, forest, fox, girl, hamster, house, kangaroo, keyboard
5	lamp, lawn mower, leopard, lion, lizard, lobster, man, maple tree, motor cycle, mountain
6	mouse, mushroom, oak tree, orange, orchid, otter, palm tree, pear, pickup truck, pine tree
7	plain, plate, poppy, porcupine, possum, rabbit, raccoon, ray, road, rocket
8	rose, sea, seal, shark, shrew, skunk, skyscraper, snail, snke, spider
9	squirrel, streetcar, sunflower, sweet pepper, table, tank, telephone, television, tiger, tractor
10	train, trout, tulip, turtle, wardrobe, whale, willow tree, wolf, woman, worm

G.4 CIFAR 10x10 Repeated Classes

We also considered the setting where each task is defined by a random sampling of 10 out of 100 classes with replacement. This environment is designed to demonstrate the effect of tasks with shared subtasks, which is a common property of real world lifelong learning tasks. Supplementary Figure 6 shows transfer efficiency of SynF and SynN on Task 1.

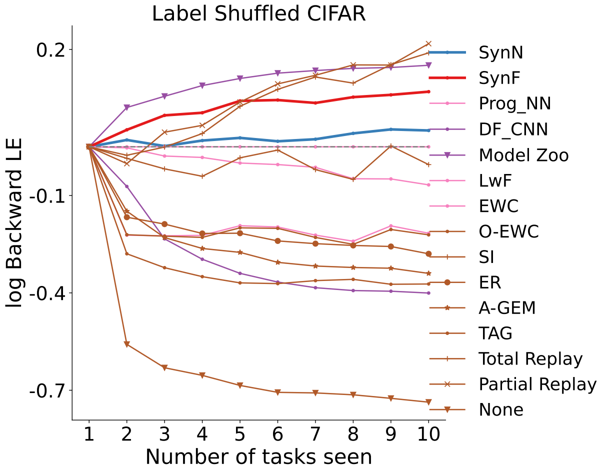

Figure 5: Label shuffle experiment on CIFAR 10x10 vision dataset for 5,000 training samples per task. Shuffling class labels within tasks two through nine with 5000 samples each demonstrates both SynF and SynN can still achieve positive backward transfer, and that the other algorithms that do not replay the previous task data fail to transfer.

Table 10: **Performance metrics: average accuracy $\langle \mathcal{A} \rangle$, forgetting $\langle \mathcal{F} \rangle$ and average transfer $\langle \mathcal{T} \rangle$ as proposed by Veniat et al. (2020) calculated for different algorithms on CIFAR 10x10 (**5000 **samples per task).**

Algorithms	$\langle \mathcal{A} \rangle$	$\langle \mathcal{F} \rangle$	$\langle \mathcal{T} \rangle$
SynN	**0.95**	**0.0**	**0.0**
SynF	**0.95**	**0.0**	**0.0**
ProgNN	0.97	0.0	0.0
LMC	0.92	0.0	0.0
DF-CNN	0.93	−0.02	−0.01
SynF(resource constrained)	**0.95**	**0.0**	**0.0**
EWC	0.95	0.0	−0.01
Total Replay	0.96	0.0	0.0
Partial Replay	0.96	0.0	0.0
Model Zoo	0.96	0.01	0.01
LwF	0.96	0.0	0.0
O-EWC	0.95	0.0	−0.01
SI	0.95	0.0	−0.01
ER	0.94	−0.02	−0.01
A-GEM	0.94	−0.02	−0.01
TAG	0.92	−0.01	−0.03
None	0.93	−0.03	−0.02

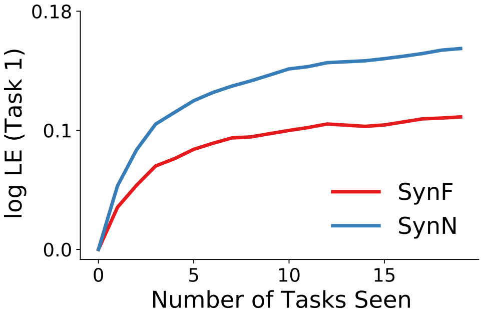

Figure 6: `SynF` and `SynN` transfer knowledge effectively when tasks share common classes. Each task is a random selection of 10 out of the 100 CIFAR-100 classes. Both `SynF` and `SynN` demonstrate monotonically increasing transfer efficiency for up to 20 tasks.