Representation Ensembling for Synergistic Lifelong Learning with Quasilinear Complexity

Joshua T. Vogelstein,[1,†] Jayanta Dey,[1,†,*] Hayden S. Helm,[1] Will LeVine,[1,5] Ronak D. Mehta,[1] Tyler M. Tomita,[1] Haoyin Xu,[1] Ali Geisa,[1] Qingyang Wang,[1] Gido M. van de Ven,[2,3] Chenyu Gao,[1] Weiwei Yang,[4] Bryan Tower,[4] Jonathan Larson,[4] Christopher M. White,[4] and Carey E. Priebe[1]

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 we define as, starts from a blank slate, or *tabula rasa* and 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 not only to improve performance on future tasks (forward transfer) but also 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 decisions (like in decision forests) and complements ensembling dependently learned representations (like in multitask learning). Moreover, we can 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, and spoken, and adversarial tasks. 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 the process by which an intelligent system improves performance on a given task by leveraging data [1]. In classical machine learning, the system is often optimized for a single task [2, 3]. While it is relatively easy to *simultaneously* optimize for multiple tasks (multitask learning) [4], it has proven much more difficult to *sequentially* optimize for multiple tasks [5, 6]. 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 [7, 8]. 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 [9].

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. [10], Zenke et al. [11], Li and Hoiem [12], Schwarz et al. [13], Finn et al. [14]. 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) [15–17]. 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. [18] where the model adaptively expands the capacity when the capacity of the model saturates.

Approaches from both camps demonstrate some degree of continual (or lifelong) learning [19]. In particular, they can sometimes learn new tasks while not catastrophically forgetting old tasks. However,

[1]Johns Hopkins University (JHU), [2]Baylor College of Medicine, [3]University of Cambridge [4]Microsoft Research, [5]Scale AI

[†] denotes equal contribution, [*] corresponding author: jdey4@jhu.edu

as we will show, many state of the art 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 [17, 20]. 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 [21, 22]. Our work falls into the (arguably simpler) resource growing camp in which each new task is learned with additional representational capacity.

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 [23] and gradient boosting tree algorithms [24]. Similarly, ensembling networks shows promising results in various real-world applications [25, 26]. In continual learning scenarios, previous works built on these ideas by ensembling dependent representations (as in ProgNN [16] and DF-CNN [17]). Our key innovation is that one can ensemble independent representations, thereby benefiting from past representations without biasing future representations. Moreover, we introduce a channel layer to enable the representations to interact with one another, thereby enabling computationally efficient forward and backward transfer.

Specifically, we introduce two complementary lifelong learning algorithms, one based on ensembling decision forests (Syngeristic Forests, SYNF), and another based on ensembling deep networks (Synergistic Networks, SYNN). Both decision forests and deep networks learn a task representation in terms of polytopes that partition the feature space [27]. SYNF and SYNN ensemble sets of polytopes learned from each task by aggregating discriminative information across tasks via a channel layer. Additionally, we propose learning metrics which are crucial in quantifying lifelong learning capabilities. We use simulation study to explore some key properties of our proposed algorithms. In our experiments, we consider a simplified learning environment akin to those previously published [10–13, 16, 17], where we know the task identities and the tasks are streaming but the data within task are batched. Moreover, when we desire to transfer backward, we keep the prior data, like replay approaches [28, 28–30]. However, previously proposed replay algorithms do not demonstrate backward transfer in our experiments. On the contrary, both SYNF and SYNN demonstrate forward and backward transfer, while maintaining computational efficiency in vision and language benchmark applications. Although the algorithms presented here are primarily resource building, we illustrate that they can effectively leverage prior representations to operate in resource constrained scenarios. This ability implies that the algorithm can convert from a “juvenile" resource building state to the “adult" resource recruiting state – all while maintaining key lifelong learning capabilities and efficiencies.

2 Background

2.1 Classical Machine Learning Classical supervised learning [31] considers random variables $(X, Y) \sim P_{X,Y}$, where X is an $\mathcal{X}$-valued input, Y is a $\mathcal{Y}$-valued label (or response), and $P_{X,Y}$ is the joint distribution of (X, Y). Given a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$, the goal is to find the hypothesis (also called predictor), $h : \mathcal{X} \to \mathcal{Y}$ that minimizes expected loss, or *risk*, $R(h) = \mathbb{E}_{X,Y}\left[\ell(h(X), Y)\right]$. A learning algorithm is a function f that maps data sets (n training samples) to a hypothesis, where a data set $\mathbf{S}_n = \{X_i, Y_i\}_{i=1}^n$ is a set of n input/response pairs. Assume n samples of (X, Y) pairs are independently and identically distributed from some true but unknown $P_{X,Y}$ [31]. A learning algorithm is evaluated on its generalization error (or expected risk): $\mathbb{E}\left[R(f(\mathbf{S}_n))\right] = \mathbb{E}\left[R(\hat{h}_n)\right]$, where the expectation is taken with respect to the true but unknown distribution governing the training data, P_{S_n}. 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 [32].

2.2 Lifelong Learning Lifelong supervised learning generalizes classical supervised machine learning in a few ways: (i) instead of one task, there is an environment $\mathcal{T}$ of (possibly infinitely) many tasks,

(ii) evaluation data-label pair (X, Y) for each task sampled from some distribution $\mathcal{D}$ arrive sequentially, rather than in batch mode, and (iii) there are computational complexity constraints on the learning algorithm and hypotheses. In supervised lifelong learning settings, one can consider the following risk for a particular task t with n random training samples $\mathbf{S}_n$ distributed as $\mathcal{D}_n$:

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

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

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

We are given the error $\mathcal{E}_f^t(\mathbf{S}_n)$ for $t = 1, \cdots, T$ and a weight for each task q_t corresponding to the extent the learner prioritizes task t such that $\sum_{t=1}^{T} q_t = 1$ and $q_t \geq 0$. Letting $\mathcal{E}_f^{\mathcal{T}}(\mathbf{S}_n) = \sum_{t\in\mathcal{T}} q_t\, \mathcal{E}_f^t(\mathbf{S}_n)$ and given a class of learners $\mathcal{F}$, the goal of a lifelong learner is to find an $f \in \mathcal{F}$ such that:

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

Implicit in the above equation is that we are not only concerned not just with past tasks, but also all possible future tasks. That said, we are not explicitly solving Objective (3) in our proposed approach.

The computational complexity constraints for lifelong learning is 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 [33], 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 is aware of all task details for all tasks, meaning that the hypotheses are of the form $h : \mathcal{X} \times \mathcal{T} \to \mathcal{Y}$. In "task-unaware" (or agnostic [34]) scenarios the learner may not know that the task has changed at all, which means that the hypotheses are of the form $h : \mathcal{X} \to \mathcal{Y}$. We only address task-aware scenarios here.

2.3 Reference algorithms We compared our approaches to 11 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 [16] and Deconvolution-Factorized CNNs (`DF-CNN`) [17] 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 5.1. `DF-CNN` [17] 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. We also compare two variants of exact replay (Total Replay and Partial Replay) [35]. Both 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 appraoch with more constrained ways of replaying old task data like- Model Zoo [36], Averaged Gradient Episodic Memory (`A-GEM`) [37], Experience Replay (`ER`) [38] and Task-based Accumulated Gradients (`TAG`) [39]

for lifelong learning. Among them Model Zoo builds on our approach and ensembles multiple representations using the boosting approach. In Model Zoo, the total number of models within the ensemble (the number of episode) 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`) [10], Online-EWC (`O-EWC`) [13], Synaptic Intelligence (`SI`) [11], Learning without Forgetting (`LwF`) [12], 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 [17, 40]; for implementation details, see Appendix D.

3 Evaluation Criteria Others have previously introduced criteria to evaluate transfer, including forward and backward transfer [41–43]. These definitions typically compare the difference, rather than the ratio, between learning with and without transfer. Pearl [44] introduced the transfer benefit ratio, which builds directly off relative efficiency from classical statistics [32]. Our definitions are closely related to Pearl's definition. In Subsection 3.1, we compare our proposed metrics with the existing ones by providing a concrete example. Here, we first formally define our metrics. *Learning efficiency* is 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. Consider a lifelong learning environment with total T tasks introduced to the learning agent sequentially. Let R^t be the risk associated with Task t, and $\mathbf{S}^i$ be the data that is specifically associated with any Task i with sample size n_i, so $R^t(f(\mathbf{S}^t))$ is the risk on Task t of the hypothesis learned by f only using Task t data, and $R^t(f(\bigcup_{i=1}^{T}\mathbf{S}^i))$ denotes the risk on Task t of the hypothesis learned on all the data up to Task T. Note that, $\sum_{i=1}^{T} n_i = n$.

Definition 1 (Learning Efficiency). *The learning efficiency of algorithm f for given Task t with total sample size n is:*

$$\mathsf{LE}_n^t(f) := \frac{\mathcal{E}_f^t(\mathbf{S}^t)}{\mathcal{E}_f^t(\bigcup_{i=1}^{T}\mathbf{S}^i)}. \tag{4}$$

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. *Forward* learning efficiency is the expected ratio of the generalization error of the learning algorithm with (i) access only to Task t data, to (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 2 (Forward Learning Efficiency). *The forward learning efficiency of f for task t given n samples is :*

$$\mathsf{FLE}_n^t(f) := \frac{\mathcal{E}_f^t(\mathbf{S}^t)}{\mathcal{E}_f^t(\bigcup_{i=1}^{t}\mathbf{S}^i)}. \tag{5}$$

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 $\mathcal{E}_f^t(\bigcup_{i=1}^t \mathbf{S}^i)$ 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 3 (Backward Learning Efficiency). *The backward learning efficiency of f for Task t given n samples is*

$$\text{BLE}_n^t(f) := \frac{\mathcal{E}_f^t(\bigcup_{i=1}^t \mathbf{S}^i)}{\mathcal{E}_f^t(\bigcup_{i=1}^T \mathbf{S}^i)}. \tag{6}$$

We say an algorithm (positively) backward transfers to Task t from all the tasks T if and only if $\text{BLE}_n^t(f) > 1$. We can report $\text{BLE}_n^t(f)$ for each t as we gradually increase the number of total task T in the environment or we can report the final $\text{BLE}_n^t(f)$ 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 $\text{BLE}_n^t(f) > 1$, then the algorithm has used data associated with future tasks to improve performance on past tasks.

After observing T tasks, the extent to which the LE for the i^{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:

$$\text{LE}_n^t(f) = \frac{\mathcal{E}_f^t(\mathbf{S}^t)}{\mathcal{E}_f^t(\bigcup_{i=1}^T \mathbf{S}^i)} = \frac{\mathcal{E}_f^t(\mathbf{S}^t)}{\mathcal{E}_f^t(\bigcup_{i=1}^t \mathbf{S}^i)} \times \frac{\mathcal{E}_f^t(\bigcup_{i=1}^t \mathbf{S}^i)}{\mathcal{E}_f^t(\bigcup_{i=1}^T \mathbf{S}^i)}.$$

Throughout, we will report log LE so that positive learning corresponds to 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,

$$\log \sum_{t \in \mathcal{T}} m_t \cdot \text{LE}_n^t(f) > 0 \tag{7}$$

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 quantity in (7) is positive for all of the possible convex combinations of all the tasks up to T.

3.1 A concrete example on lifelong learning metrics In this section, we propose that it is desirable for lifelong learning metrics to have a few properties that are not currently present in the existing metrics. First, the amount of overall transfer should be decomposable into forward and backward components. This enables one to report an overall level of transfer if desired as a summary, rather than requiring yet a third metric to quantify the amount of overall transfer. Please see figure 4 last row first panel where we report final learning efficiency after 10 tasks as a summary of the overall performance of the model.

Second, one should be able to generally define transfer learning, and then obtain forward and backward as two specific interesting cases. This is because both forward and backward transfer are special cases of transfer learning achieved from two different streams of data, i.e., past task data and future task data, respectively. In our proposed metrics, both BLE and FLE are defined by the same exact function, just with different data streams in the numerator and the denominator.

Third, the amount of transfer should be dependent on the accuracy level of the algorithms. This is because in general, once we get to high accuracy levels (e.g., 98% or so), we care deeply about gains in *relative* performance, that is, reducing error from 2% to 1% is a big deal. In contrast, if one reduces error from 49% to 48%, that is relatively less interesting and impactful. Recall that for Bernoulli random variables, the variance of the estimator is a function of how close it is to 50%. In both of the cases, the change in accuracy is 1% whereas learning efficiencies (LE) are 2 and 1.02, respectively. Therefore, our proposed metrics automatically account for the relative difficulties of transfer learning at different accuracy levels.

Fourth, the metrics should be able to resolve the overall transfer learning into individual tasks. For example, while reporting an average overall transfer a task with extremely high transfer may mask the poor performance over the other tasks.

Fifth, the metrics should validly quantify the amount of actual *transfer* of information from one (set of) data to another, rather than merely improvement. The overall accuracy over the tasks can improve simply because subsequent tasks are easier, for example. This improvement in accuracy does not indicate whether there has been any transfer. We will elaborate the aforementioned phenomenon further on a benchmark dataset in Figure 3. The metrics that we propose here satisfy all five of the above desiderata; we show below that existing metrics do not satisfy some of them.

The following example justifies how our proposed metrics satisfy all the five aforementioned desiderata. We compare and contrast our proposed metrics with the metrics proposed in Díaz-Rodríguez et al. [43] on a hypothetical scenario, including accuracy, backward transfer (BWT), and forward transfer (FWT). Consider a lifelong learning environment with two tasks each having two classes. The tasks are introduced sequentially with n_1 samples from Task 1 and then n_2 samples from Task 2. The agent has a generalization error of $R^1(f(S^1)) = 0.3$ on Task 1 while it has access to the Task 1 dataset only, and a generalization error of $R^2(f(S^2)) = 0.4$ on Task 2, while it has access to the Task 2 dataset only. Now consider the scenario when the agent has the same hyper-parameters and sequential access to all the task datasets. Suppose the model has the generalization error on two tasks enumerated as in table 1. Note that the FLEs are given by: $\mathsf{FLE}^1_n(f) = \frac{R^1(f(\mathbf{S}^1))}{R^1(f(\mathbf{S}^1))} = 1$ and $\mathsf{FLE}^2_n(f) = \frac{R^2(f(\mathbf{S}^2))}{R^2(f(\mathbf{S}^1 \bigcup \mathbf{S}^2))} = 0.89$. The performance metrics can be summarized as in table 1.

Table 1: Learning metrics summarized on a hypothetical scenario.

Metrics	Dataset	
	$S = S^1$	$S = S^1 \bigcup S^2$
$R^1(f(S))$	0.30	0.32
$R^2(f(S))$	0.5	0.45
$\mathsf{BLE}^1_n(f)$	1	0.94
$\mathsf{FLE}^1_n(f)$	1	1
$\mathsf{LE}^1_n(f) = \mathsf{BLE}^1_n(f) \times \mathsf{FLE}^1_n(f)$	1	0.94
$\mathsf{BLE}^2_n(f)$	$-$	1
$\mathsf{FLE}^2_n(f)$	$-$	0.89
$\mathsf{LE}^2_n(f) = \mathsf{BLE}^2_n(f) \times \mathsf{FLE}^2_n(f)$	$-$	0.89
Average global accuracy	0.60	0.62
BWT	0	-0.02
FWT	0	0.5

As evident in Table 1, the transfer learning for Task 1 comes from backward transfer from Task 2 whereas for Task 2 it comes from forward learning from Task 1. As a summary, one can look at the final LEs over all the tasks after all tasks have been introduced. Note that in Table 1 the learning efficiencies are never greater than 1. However, the global accuracy increased from 60% to 62%. Therefore, only

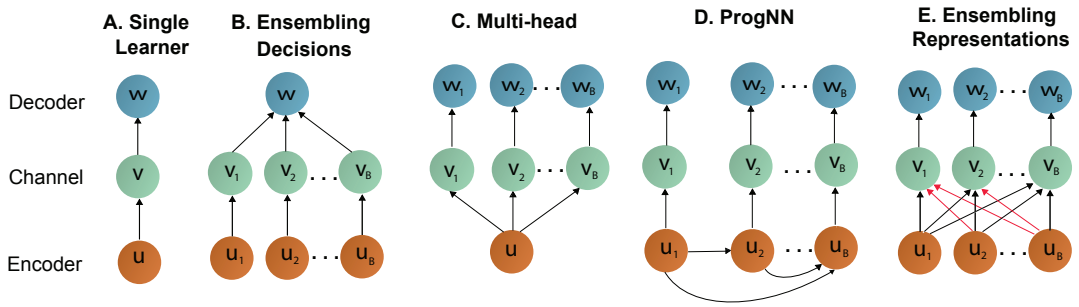

Figure 1: Schemas of composable hypotheses. A. Single task learner. B. Ensembling decisions (as output by the channels) is a well-established practice, including random forests and gradient boosted trees. C. Learning a joint representation or D. Ensembling representations (learned by the encoders) was 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.

using multi-task accuracy may falsely detect positive transfer. In Table 1, BWT can correctly identify an overall negative backward transfer or forgetting. However, being an average quantity, it can not resolve the overall backward transfer into individual task. As a result, a task with extremely high backward transfer may mask all the negative backward transfer from the other tasks giving a net positive transfer.

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 and 4.2. Our approach to lifelong learning is based on decomposition of the hypothesis learned by a model into an encoder, a channel, and a decoder [45, 46] (Figure 1A): $h(\cdot) = w \circ v \circ u(\cdot)$. The encoder, $u : \mathcal{X} \mapsto \tilde{\mathcal{X}}$, maps an $\mathcal{X}$-valued input into an internal representation space $\tilde{\mathcal{X}}$ [47, 48]. The channel $v : \tilde{\mathcal{X}} \mapsto \Delta_{\mathcal{Y}}$ maps the transformed data into a posterior distribution (or, more generally, a score). For example, consider we have a dataset partitioned into a training and a held-out set. Now we can learn a decision tree using the training data which will give us the encoder. Next, by pushing the held-out dataset through the tree, we can learn the channel, i.e., posteriors in the leaf-nodes. The channel thus gives scores for each data point denoting the probability of that data point belonging to a specific class. Finally, a decoder $w : \Delta_{\mathcal{Y}} \mapsto \mathcal{Y}$, produces a predicted label. See Appendix A for a detailed and concrete example using a decision tree.

One can generalize the above decomposition by allowing for multiple encoders. Given B different encoders, one can attach a single channel to each encoder, yielding B different channels (Figure 1B). Doing so requires generalizing the definition of a decoder, which 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 boosting [49] and bagging [50]—indeed, decision forests are a canonical example of a decision function operating on a collection of B outputs [23]. 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 1B 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, for example. Therefore, in the multi-head structure (Figure 1C) 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 [10, 11, 13]. Further modification of the multi-head structure allows ProgNN to learn separate encoder for each task with forward connections from the past encoders to the current one (Figure 1D). 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. Thus, a generalization of the decomposition in Figure 1B allows for each channel to ensemble the encoders (Figure 1E). Doing so requires generalizing the definition of the *channel*, so that it can operate on multiple distinct encoders. The result is that the channels **ensemble representations** (learned by the encoders), rather than decisions (learned by the channels). The channels ensemble all the existing representations, regardless of the order in which they were learned. In this scenario, like with bagging and boosting, the ensemble of channels then feeds into the single decoder. When each encoder has learned complementary representations, this latter approach has certain appealing properties, particularly in multiple task scenarios, including lifelong learning.

For example, Model Zoo [36] which builds on our approach ensembles multiple encoders learned over different subsets of tasks using the boosting approach. On the other hand, we developed two different representation ensembling algorithms based on bagging of models trained on individual task. As we will show empirically, these two ensemble methods tend to outperform the existing state-of-the-art algorithms. It is shown in Wyner et al. [51] that both bagging and boosting asymptotically converges 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 4) whereas boosting performs better on large training sample size (See main text Figure 6, 7 and Appendix Figure 2).

The key to both of our algorithms is the realization that both forests and networks partition feature space into a union of polytopes [27]. 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 and B for details).

In either of the algorithms, as new data from a new task arrives, our algorithm first builds a new independent encoder, i.e., a set of polytopes (using forests or networks), mapping each data point to a sparse vector encoding which polytope it is in. Then, it builds the channel for this new task, which integrates information across all existing encoders using the new task data, thereby enabling forward transfer. At the same time, 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.

4.1 Synergistic Forests Synergistic Forests (SYNF) ensembles decision trees or forests. For each task, the encoder u_t of a SYNF is the representation learned by a decision forest [23, 52]. The leaf nodes of each task corresponding decision forest partition the input space $\mathcal{X}$ into polytopes [53]. 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" [53–55]. 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 [55], 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 approaches simplifies to an approach called "Uncertainty Forests" [56]. Table 1 in the appendix lists the hyperparameters used in the CIFAR experiments.

4.2 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) [57] 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 [57]. 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.* SYNF adds yet another difference as compared to SYNN by replacing the deep network encoders with random forest encoders. This has the effect of making the capacity, space complexity, and time complexity scale with the complexity and sample size of each task. In contrast, both ProgNNand SYNN have a fixed capacity for each task, even if the tasks have very different sample sizes and complexities.

5 Results

5.1 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 complexity of the internal representation of a number of algorithms. We also study the representation capacity of these algorithms. We use the soft-O notation $\tilde{\mathcal{O}}$ to quantify complexity [58]. Letting n be the sample size and T be the number of tasks, we write that 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. Again, while calculating the space complexity, we have ignored the space required for growing a new head for the new task. Table 2 summarizes the capacity, space and time complexity of several reference algorithms, as well as our SYNN and SYNF. For the deep learning methods, we assume that the number of iterations is proportional to the number of samples. 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, $n \propto T$.

Table 2: Capacity, space, and time constraints 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. Our algorithms and DF-CNN are the only 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
non-parametric	n	n	n	n	n	SYNF

Parametric lifelong learning methods have a representational capacity is invariant to sample size and task number. Although the space complexity of some of these algorithms grow (because the size of the constraints grows, or they continue to store more and more data), their capacity is fixed. Thus, given a sufficiently large number of tasks, without placing constraints on the relationship between the tasks, eventually all parametric methods will catastrophically forget at least some things. EWC, ONLINE

EWC, SI, and LwF 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 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 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 eliminates the lateral connections between columns of the network, thereby reducing space complexity down to $\tilde{\mathcal{O}}(T)$. SYNN stores all the data to enable backward transfer, but retains linear time complexity.

SYNF is the only non-parametric lifelong learning algorithm to our knowledge. Its capacity, space and time complexity are all $\tilde{\mathcal{O}}(n)$, meaning that its representational capacity naturally increases with the complexity of each task.

5.2 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.

Synergistic learning in a simple environment Consider a very simple two-task environment: Gaussian XOR and Gaussian Exclusive NOR (XNOR) (Figure 2A, see Appendix E 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 2Bi). 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 2Bii). 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 2Biii). 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. Qualitatively similar results are visible for SYNN and DN in Figure 2.

Synergistic learning in adversarial environments Statistics has a rich history of *robust learning* [59, 60], and machine learning has recently focused on *adversarial learning* [61–64]. 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 2Aiii). Training on R-XOR therefore impedes the

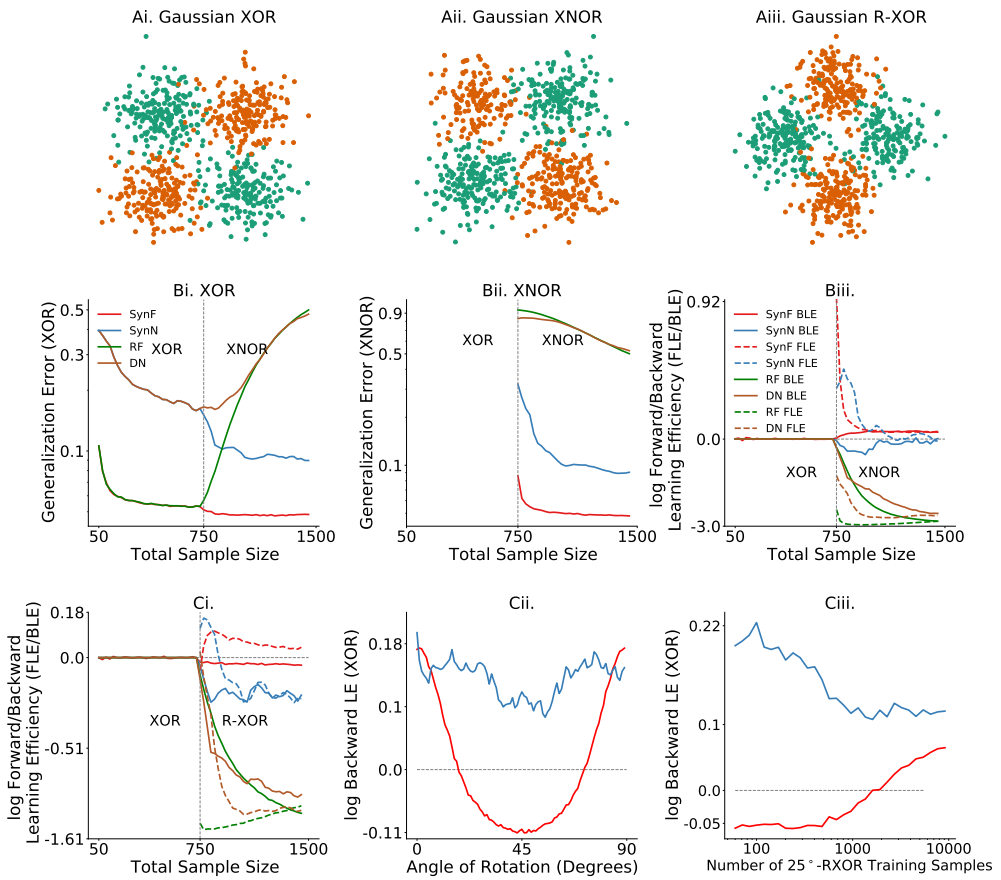

Figure 2: **Synergistic Forest and Synergistic Network demonstrate forward and backward transfer.** The learner is trained from scratch for each sample size so that we can observe the impact of increasing sample size on our algorithms. (*A*) 750 samples from: (*Ai*) Gaussian XOR, (*Aii*) XNOR, which 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 sand DN. (*Biii*) Forward and backward learning efficiency of SYNF are positive for all sample sizes, and are negative for all sample sizes for RF. Again, FLE and BLE is higher for SYNNcompared to those of DN. (*Ci*) In an adversarial task setting (100 samples of XOR followed by 100 samples of R-XOR), SYNFand SYNN gracefully forgets XOR, whereas RFand DN demonstrate catastrophic forgetting and interference. (*Cii*) log BLE with respect to XOR is positive 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.

performance of SYNF and SYNN on XOR, and thus backward transfer becomes negative, demonstrating graceful forgetting [65] (Figure 2Ci). Because R-XOR is more difficult than XOR for SYNF (because the discriminant boundaries are oblique [66]), 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 2Cii). 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 2Ciii).

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 E further investigates this phenomenon in a multi-spiral environment.

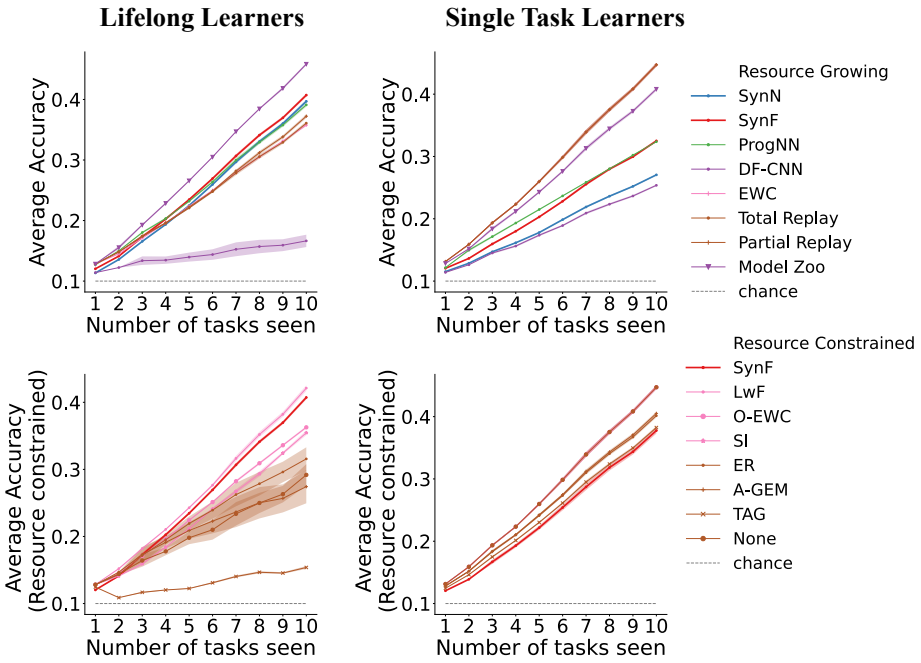

Figure 3: **Average accuracy over 10 tasks as the learners (lifelong and single task learners) see more tasks.** For single task learners, a new stand-alone learner is trained as a new task is seen and the average accuracy over all the task specific learners is reported. LwF has the highest multitask 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.

5.3 Benchmark data experiments For benchmark data, we build SYNN encoders using the network architecture described in van de Ven et al. [28] 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 as well. For the following experiments, we consider two modalities of real data: vision and language. Our language experiments have qualitatively similar results as those of vision experiments illustrating that SYNF and SYNN are modality agnostic, sample and computationally efficient lifelong learning algorithms. In addition to the CIFAR 100 dataset, we provide additional vision experiments on larger datasets which show the relative performance gain of Model Zoo (boosting) compared to that of our approach (bagging) on large datasets. 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 [17, 20], 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 2.

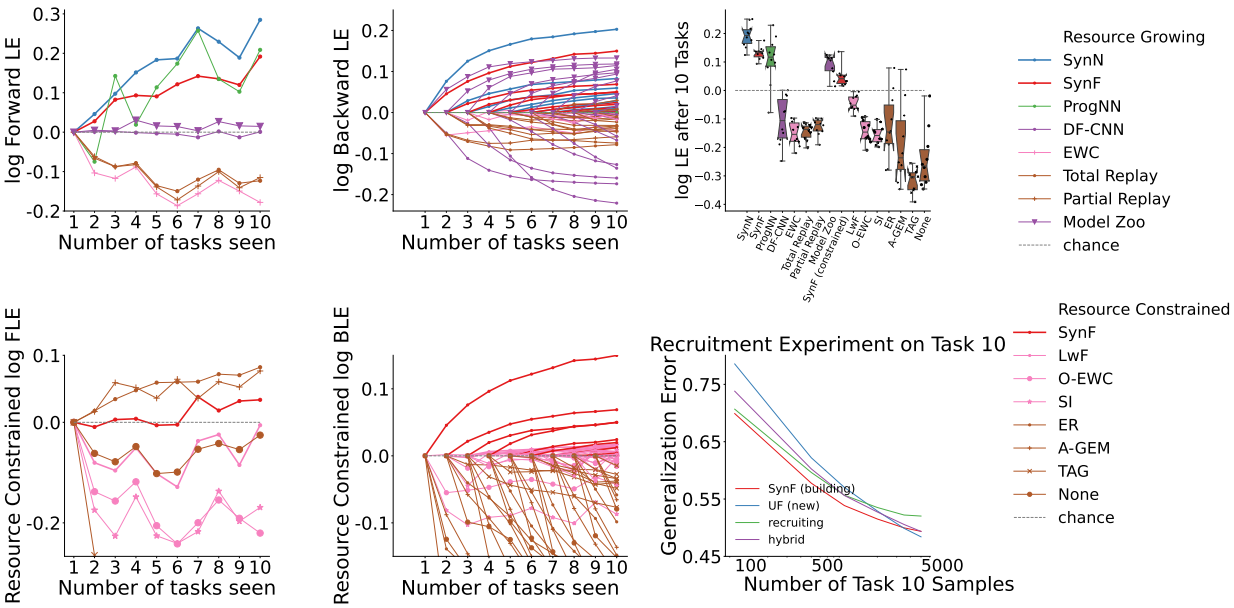

Figure 4: **Performance of different algorithms on the CIFAR 10x10 vision experiments.** *Top left and middle*: Forward and backward transfer efficiency for various resource building algorithms. SYNF and SYNN consistently demonstrate both forward and backward transfer for each task, whereas ProgNN and DF-CNN 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. *Bottom left and middle*: Same as above but comparing each algorithm with a fixed amount of resources. SYNF is the only approach that demonstrate forward or backward transfer. *Top right*: Transfer efficiencies of various algorithms for the 10 tasks after seeing the 10-th task. Both SYNN and SYNF synergistically learn over all the 10 tasks whereas other algorithms (except ProgNN) catastrophically forget. *Bottom right*: Building and recruiting 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.

CIFAR 10x10 dataset The CIFAR 100 challenge [67], 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 10x10 divides these data into 10 tasks, each with 10 classes [17] (see Appendix F for details). We compare SYNF and SYNN to the deep lifelong learning algorithms discussed above. In the subsequent experiments, we have reported the average accuracy over all the tasks as more tasks are seen as proposed in [68, 69] for both lifelong and single task learners along with our proposed learning efficiencies. However, only multitask accuracy cannot ascertain the superiority of an algorithm. For example, note that in Figure 3 bottom left, LWF has better average global accuracy compared to that of SYNF. However, as shown in the bottom right of Figure 3, LWF has relatively higher single task accuracy compared to that of SYNF, i.e., accuracy when the learner has access to a single task data only. 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 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 $\log(\text{BLE})$ curve starts from 0 when the corresponding task is introduced and goes upward (positive) or downward (negative) as more tasks are seen.

Resource growing experiments We first compare SYNF and SYNN to state-of-the-art resource growing algorithms: Model Zoo, ProgNN and DF-CNN (Figure 4, top panels). Both SYNF and SYNN demon-

strate positive forward transfer for every task (SYNF increases nearly monotonically), indicating they are robust to distributional shift in ways that ProgNNand DF-CNN are not. SYNN, SYNF and Model Zoo 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, neither ProgNN nor DF-CNN exhibit any positive backward transfer. Final learning efficiency per task in the third row first plot is the learning efficiency associated with that task having seen all the data. SYNF and SYNN both demonstrate positive final learning efficiency for all tasks (synergistic learning), whereas ProgNN and DF-CNN both exhibit negative final learning efficiency for at least one task.

Resource constrained experiments It is possible that the above algorithms are leveraging additional resources to improve performance without meaningfully transferring information between representations. To address this concern, we devised a "resource constrained" variant of 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). If SYNF is able to meaningfully transfer information across tasks, then its resource-constrained FLE and BLE will still be positive. Indeed, FLE remains positive after enough tasks, and BLE is actually invariant to this change (Figure 4, bottom left and center). In contrast, all of the reference algorithms that have 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 4, top right). Interestingly, when using 5,000 samples per task, total and partial replay methods are able to demonstrate positive forward and backward transfer (Supplementary Figure 5), 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 omnidirectional learners requires $\tilde{\mathcal{O}}(n^2)$ resources, where as SYNF only requires $\tilde{\mathcal{O}}(n)$, a polynomial reduction in resources.

In both cases, resource growing or resource constrained, both SYNF and SYNN show synergistic learning over all the 10 tasks (Figure 4, top right panel) whereas all other algorithms except ProgNNsuffer from catastrophic forgetting.

Resource Recruiting 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 (juvenile) to constrained (adult) resources (which requires recruiting some resources for new tasks). We therefore train 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 (recruiting), (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 recruiting approach is nearly as good as SYNF (Figure 4, bottom 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.

Adversarial experiments Consider the same CIFAR 10x10 experiment above, but, for tasks two through nine, 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 5A indicates that BLE 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

Table 3: 5-dataset details.

	Training samples	Testing samples
CIFAR-10	50000	10000
MNIST	60000	10000
SVHN	73257	26032
notMNSIT	16853	1873
Fashion-MNIST	60000	10000

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 5, 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*.

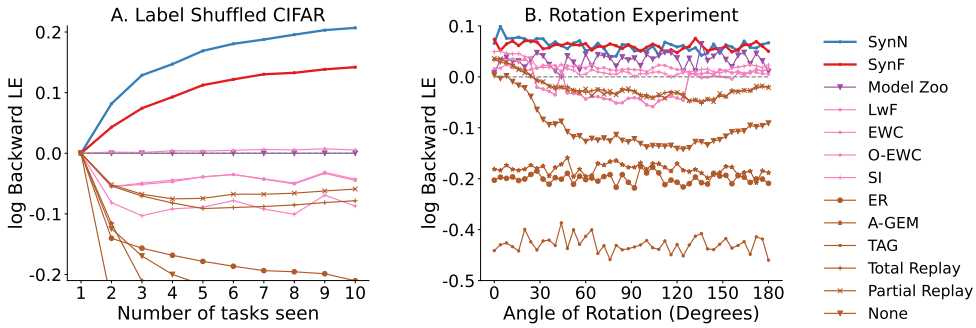

Figure 5: **Extended CIFAR 10x10 experiments.** *A.* Shuffling class labels within tasks two through nine with 500 samples each demonstrates both SYNF and SYNN can still achieve positive backward transfer, and that the other algorithms still fail to transfer. *B.* SYNF and SYNN are nearly invariant to rotations, whereas other approaches are more sensitive to rotation.

Five Dataset In this experiment, we have used the 5-dataset provided in https://github.com/pranshu28/TAG. It consists of 5 tasks from five different dataset- CIFAR-10 [67], MNIST, SVHN [70], notMNIST [71], Fashion-MNIST [72]. All the monochromatic images are converted to RGB format depending on the dataset. All images are then resized to $3 \times 32 \times 32$. As shown in table 3, training samples per task in 5-dataset is relatively higher than that of low data regime ideally considered in lifelong learning setting. However, SYNN and SYNFshow less forgetting than most of the benchmarking algorithms. On the other hand, model zoo shows comparatively better performance in relatively high task data size setup.

Split Mini-Imagenet In this experiment, we have used the mini-imagenet dataset 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. In this case, we get positive FLE and BLE for both SYNN and SYNF. However, model zoo outperforms all the algorithms in this experiment.

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 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 8. 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

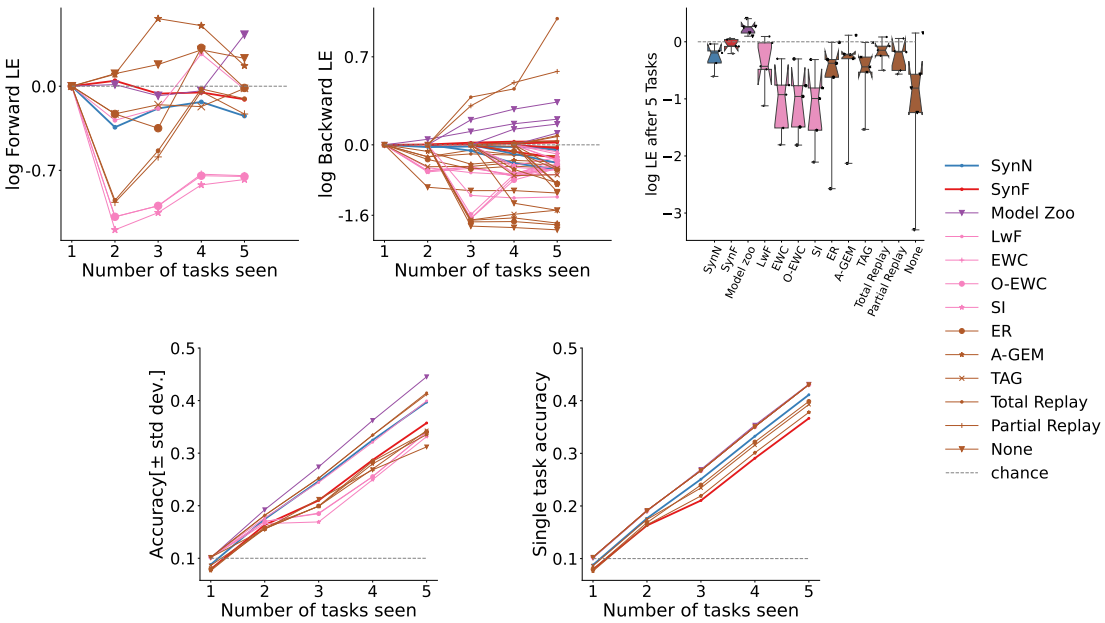

Figure 6: Model Zoo performs the best and SYNF performs the second best compared to all other algorithms in high sample size regimes (first row third panel). Sample size for each task is provided in Table 3.

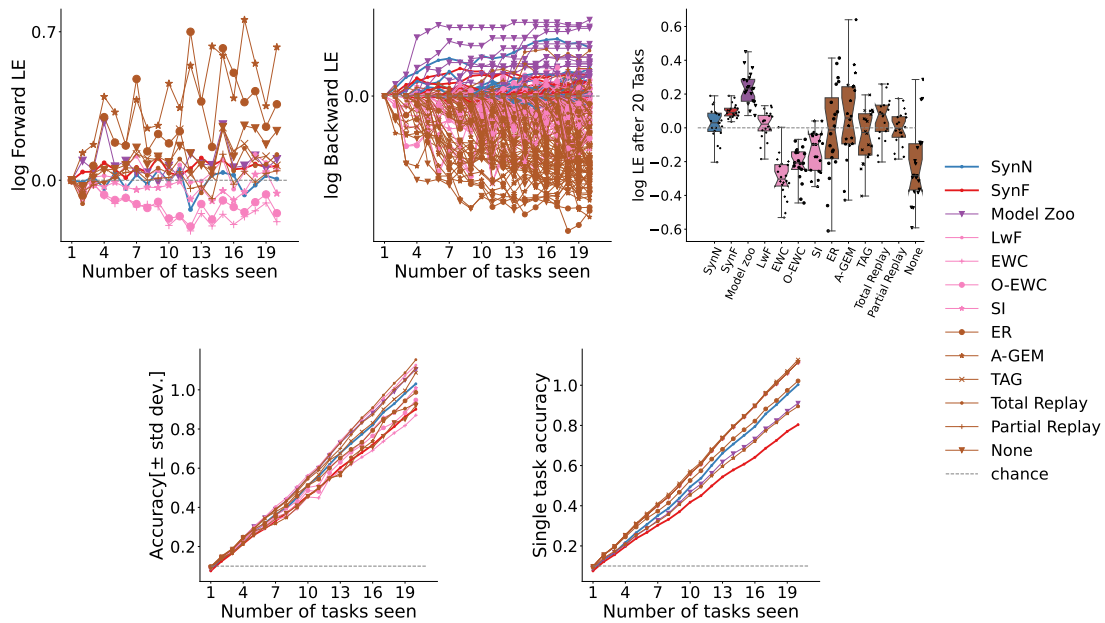

Figure 7: We have qulitatively similar results on Split Mini-Imagenet tasks as those of Five Dataset tasks. Note that each task in Mini-Imagnet has 2400 training samples which is lower than that of Five Dataset tasks. This relatively lower sample size results in a bit better performance for SYNF and SYNN compared to those on Five Dataset (first row third panel).

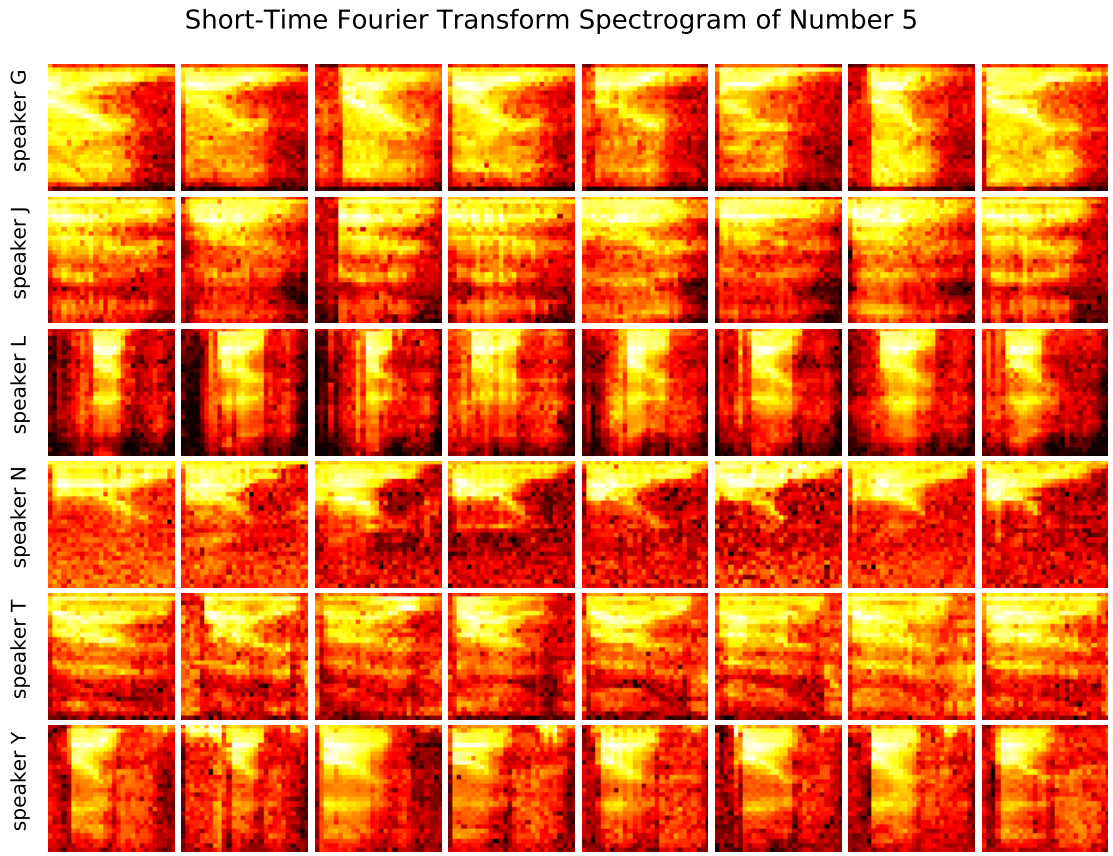

Figure 8: Spectrogram extracted from 8 different recordings of 6 speakers uttering the digit '5'.

mentioned in Subsection 2.3. As shown in Figure 9, 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.

6 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. Recruitment 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 [73].

To achieve backward transfer, SYNF and SYNN stored old data to vote on 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 (TE $= 1$) 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. For example, both ProgNN and Total Replay have quadratic time complexity, unlike SYNF and SYNN. Thus, one natural extension of this work that could mitigate the need to store all the data by using a generative model or subsampling.

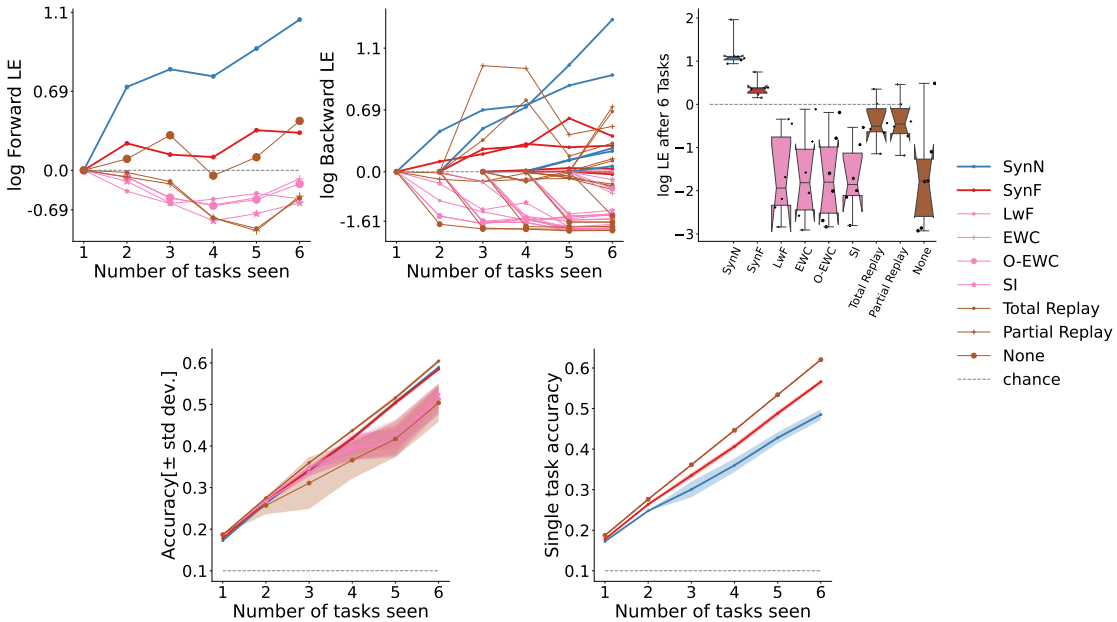

Figure 9: Both SYNF and SYNN show positive forward and backward transfer as well as synergistic learning for the spoken digit tasks, in contrast to other seven methods, some of which show only forward transfer, others show only backward transfer, with none showing both, and some showing neither.

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 [74] and genomics [75]. Similarly, federated learning is becoming increasingly central in artificial intelligence, due to its importance in differential privacy [76]. 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 [77].

Finally, our quasilinear representation ensembling approach closely resembles the constructivist view of brain development [78, 79]. 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 a proof). However, there is also substantial pruning during development and maturity in the brain circuitry which is important for performance [80]. This motivates future work for pruning adversarial 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 synergistically learn so efficiently, and transfer that understanding to building more effective artificial intelligences. In the meantime, our code, including code to reproduce the experiments in this manuscript, is available from http://proglearn.neurodata.io/.

Acknowledgements The authors thank the support of the NSF-Simons Research Collaborations on the Mathematical and Scientific Foundations of Deep Learning (NSSF grant 2031985). We also thank

Raman Arora, Dinesh Jayaraman, Rene Vidal, Jeremias Sulam, Guillermo Sapiro, and Michael Powell for helpful discussions. This work is graciously supported by the Defense Advanced Research Projects Agency (DARPA) Lifelong Learning Machines program through contracts FA8650-18-2-7834 and HR0011-18-2-0025. Research was partially supported by funding from Microsoft Research and the Kavli Neuroscience Discovery Institute.

References

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

[2] 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.

[3] 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.

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

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

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

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

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

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

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

[16] 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.
[17] 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, pages 2837–2844, 2019.
[18] Shagun Sodhani, Sarath Chandar, and Yoshua Bengio. Toward training recurrent neural networks for lifelong learning. Neural computation, 32(1):1–35, 2020.
[19] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. Neural Networks, 2019.
[20] 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.
[21] Judea Pearl. The seven tools of causal inference, with reflections on machine learning. Commun. ACM, February 2019.
[22] Gary Marcus and Ernest Davis. Rebooting AI: Building Artificial Intelligence We Can Trust. Pantheon, September 2019.
[23] Leo Breiman. Random forests. Machine learning, 45(1):5–32, 2001.
[24] 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, pages 785–794, New York, NY, USA, August 2016. Association for Computing Machinery.
[25] 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), pages 1–6. IEEE, 2014.
[26] 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), pages 621–624. IEEE, 2016.
[27] Carey E Priebe, Joshua T Vogelstein, Florian Engert, and Christopher M White. Modern Machine Learning: Partition & Vote. September 2020.
[28] 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.
[29] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. Connection Science, 7 (2):123–146, 1995.
[30] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In Advances in Neural Information Processing Systems, pages 2990–2999, 2017.
[31] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. Foundations of Machine Learning. MIT Press, November 2018. URL https://market.android.com/details?id=book-dWB9DwAAQBAJ.
[32] Peter J Bickel and Kjell A Doksum. Mathematical statistics: basic ideas and selected topics, volumes I-II package. Chapman and Hall/CRC, 2015.
[33] Nicolo Cesa-Bianchi and Gabor Lugosi. Prediction, Learning, and Games. Cambridge University Press, March 2006.
[34] Chen Zeno, Itay Golan, Elad Hoffer, and Daniel Soudry. Task Agnostic Continual Learning Using Online Variational Bayes. arXiv, March 2018.
[35] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In Advances in Neural Information Processing Systems, pages 350–360, 2019.
[36] Rahul Ramesh and Pratik Chaudhari. Model zoo: A growing brain that learns continually. In International Conference on Learning Representations, 2021.
[37] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. arXiv preprint arXiv:1812.00420, 2018.

[38] 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.
[39] Pranshu Malviya, Sarath Chandar, and Balaraman Ravindran. Tag: Task-based accumulated gradients for lifelong learning. arXiv preprint arXiv:2105.05155, 2021.
[40] 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.
[41] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In NIPS, 2017.
[42] Diana Benavides-Prado, Yun Sing Koh, and Patricia Riddle. Measuring Cumulative Gain of Knowledgeable Lifelong Learners. In NeurIPS Continual Learning Workshop, pages 1–8, 2018.
[43] 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.
[44] Pearl Judea. What is gained from past learning. Journal of Causal Inference, 6(1), 2018.
[45] Thomas M Cover and Joy A Thomas. Elements of Information Theory. John Wiley & Sons, New York, November 2012.
[46] 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.
[47] 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, editors, Advances in Neural Information Processing Systems 30, pages 5998–6008. Curran Associates, Inc., 2017.
[48] 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.
[49] Y Freund. Boosting a Weak Learning Algorithm by Majority. Inform. and Comput., 121(2):256–285, September 1995.
[50] Leo Breiman. Bagging predictors. Mach. Learn., 24(2):123–140, August 1996.
[51] 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.
[52] Yali Amit and Donald Geman. Shape Quantization and Recognition with Randomized Trees. Neural Comput., 9(7):1545–1588, October 1997.
[53] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. Classification and regression trees. CRC press, 1984.
[54] 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, editors, Proceedings of the 31st International Conference on Machine Learning, volume 32 of Proceedings of Machine Learning Research, pages 665–673, 6 2014.
[55] S. Athey, J. Tibshirani, and S. Wager. Generalized random forests. Annals of Statistics, 47(2): 1148–1178, 2019.
[56] Ronak Mehta, Richard Guo, Cencheng Shen, and Joshua Vogelstein. Estimating information-theoretic quantities with random forests. arXiv preprint arXiv:1907.00325, 2019.
[57] Charles J Stone. Consistent Nonparametric Regression. Ann. Stat., 5(4):595–620, July 1977.
[58] 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.

[59] Peter J Huber. Robust statistical procedures, volume 68. Siam, 1996.
[60] Marco Ramoni and Paola Sebastiani. Robust learning with missing data. Machine Learning, 45 (2):147–170, 2001.
[61] 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.
[62] 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, pages 335–340, 2018.
[63] 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, pages 11278–11287. PMLR, 2020.
[64] Daniel Lowd and Christopher Meek. Adversarial learning. In Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, pages 641–647, 2005.
[65] 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, editors, Computer Vision – ECCV 2018, pages 144–161, Cham, 2018. Springer International Publishing.
[66] 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.
[67] Alex Krizhevsky. Learning multiple layers of features from tiny images. University of Toronto, 05 2012.
[68] Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In Conference on Robot Learning, pages 17–26. PMLR, 2017.
[69] Davide Maltoni and Vincenzo Lomonaco. Continuous learning in single-incremental-task scenarios. Neural Networks, 116:56–73, 2019.
[70] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.
[71] Yaroslav Bulatov. http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html. 2011.
[72] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747, 2017.
[73] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong Learning with Dynamically Expandable Networks. International Conference on Learning Representations, August 2017.
[74] E W Bridgeford, S Wang, Z Yang, Z Wang, T Xu, and others. Big Data Reproducibility: Applications in Brain Imaging. bioRxiv, 2020.
[75] 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.
[76] Cynthia Dwork. Differential Privacy: A Survey of Results. In Theory and Applications of Models of Computation, pages 1–19. Springer Berlin Heidelberg, 2008.
[77] 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.
[78] Steven R Quartz. The constructivist brain. Trends in cognitive sciences, 3(2):48–57, 1999.
[79] Annette Karmiloff-Smith. From constructivism to neuroconstructivism: The activity-dependent structuring of the human brain. In After Piaget, pages 1–14. Routledge, 2017.

[80] 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.

[81] 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.

Appendix 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.

Appendix 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 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 [16, 81].

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 [15]. 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.

Appendix 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 detais of how to update representers and voters, 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 representers, updating voters, and making predictions for any SYNX algorithm; the below sections provide SYNF and SYNN specific details.

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

Input:
(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
Output:
(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

Algorithm 2 Add a new SYNX channel for the current task.

Input:
(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
Output: $\boldsymbol{v}_t = \{v_{t,t'}\}_{t'=1}^t$ ▷ 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

Appendix D. Reference Algorithm Implementation Details. The same network architecture was used for all compared deep learning methods. Following van de Ven et al. [28], 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. Each column has an input layer followed by 4 convolutional layer with size $3 \times 3 \times 32$, $3 \times 3 \times 32$, $3 \times 3 \times 64$ and $3 \times 3 \times 64$, respectively. It is followed by a fully-connected layer with 64 nodes and an output layer with 10 nodes. ReLU activation was used after each layer. The other algorithms use a common architecture with input layers defined by the size of the input data, two hidden layers with 400 nodes each and a multi-headed output layer (different output layers for different tasks). Different algorithms only differ in the way they penalize the update of network parameters for the current task based on the previous tasks. Each of these algorithms has 1.4M parameters in total.

Appendix E. 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

Algorithm 3 Update SYNX channel for the previous tasks.

Input:
(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 $t' = 1, \cdots, t-1$
Output: $\boldsymbol{v} = \{\boldsymbol{v}_{t'}\}_{t'=1}^{t-1}$ ▷ 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

Algorithm 4 Predicting a class label using SYNX.

Input:
(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
Output: $\hat{y}$ ▷ 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
  $\hat{\mathbf{p}}_t \leftarrow \hat{\mathbf{p}}_t / T$
  $\hat{y} = \text{argmax}_i(\hat{\mathbf{p}}_t)$ ▷ find the index i of the elements in the vector $\hat{\mathbf{p}}_t$ with maximum probability
  return $\hat{y}$
end function

to evaluate the algorithms. We fit a random forest (RF) (technically, an uncertainty forest which is an honest forest with a finite-sample correction [56]) and a SYNF. We repeat this process 30 times to obtain errorbars. Errorbars in all cases were negligible.

E.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.

E.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

Table 1: Hyperparameters for SYNF in CIFAR 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

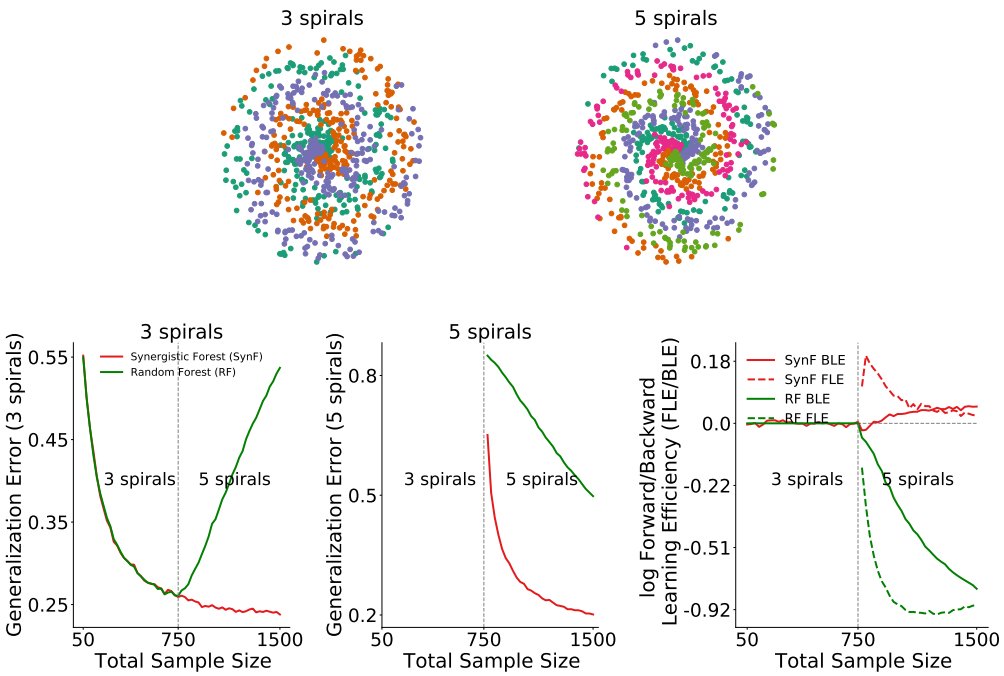

Figure 1: *Top*: 750 samples from 3 spirals (left) and 5 spirals (right). *Bottom left*: SYNF outperforms RF on 3 spirals when 5 spirals data is available, demonstrating *backward* transfer in SYNF. *Bottom center*: SYNF outperforms RF on 5 spirals when 3 spirals data is available, demonstrating *forward* 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.

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.

Appendix F. Real Data Extended Results.

F.1 CIFAR 10x10 Supplementary Table 3 shows the image classes associated with each task number. Supplementary Figure 2 is the same as Figure 4 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

Table 2: 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 3: 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

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.

F.2 CIFAR Label Shuffling Supplementary Figure 3 shows the same result as the label shuffling from Figure 5, 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.

F.3 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 4 shows transfer efficiency of SYNF and SYNN on Task 1.

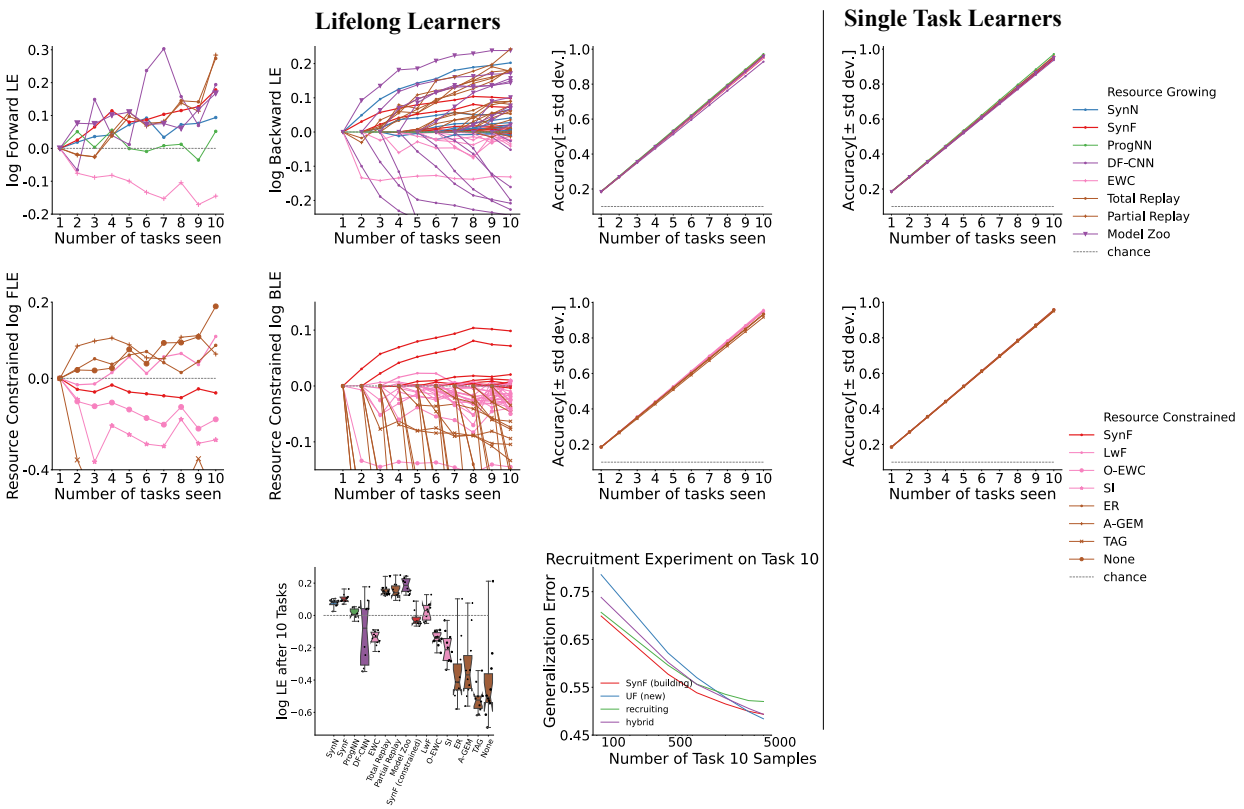

Figure 2: Performance of different algorithms on CIFAR 10x10 vision dataset for $5{,}000$ training samples per task. SYNN maintains approximately the same forward transfer (top left and middle left) and backward transfer (top and middle row second column) efficiency as those for 500 samples per task 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.

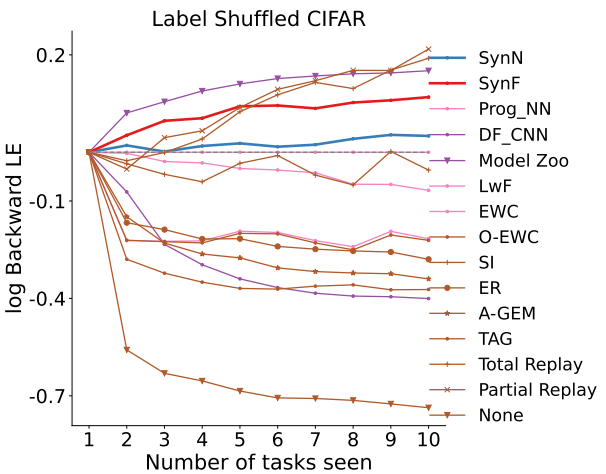

Figure 3: Label shuffle experiment on CIFAR 10x10 vision dataset for $5{,}000$ training samples per task. Shuffling class labels within tasks two through nine with 5000 samples each demonstrates both SYNF and SYNN can still achieve positive backward transfer, and that the other algorithms that do not replay the previous task data fail to transfer.

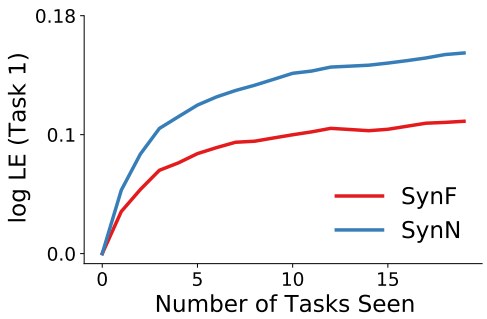

Figure 4: SYNF and SYNN transfer knowledge effectively when tasks share common classes. Each task is a random selection of 10 out of the 100 CIFAR-100 classes. Both SYNF and SYNN demonstrate monotonically increasing transfer efficiency for up to 20 tasks.