Representation Ensembling for Synergistic Lifelong Learning with Quasilinear Complexity

Jayanta Dey (jdey4@jhmi.edu)
Johns Hopkins School of Medicine https://orcid.org/0000-0002-6713-7402
Joshua Vogelstein
Johns Hopkins University
Hayden Helm
Johns Hopkins University
Will Levine
Johns Hopkins University
Ronak Mehta
Johns Hopkins University
Ali Geisa
Johns Hopkins University
Haoyin Xu
Johns Hopkins University https://orcid.org/0000-0001-8235-4950
Gido van de Ven
Baylor College of Medicine https://orcid.org/0000-0002-5239-5660
Emily Chang
Johns Hopkins University
Chenyu Gao
Johns Hopkins University
Weiwei Yang
Microsoft Research
Bryan Tower
Microsoft Research
Jonathan Larson
Microsoft Research
Christopher White
Microsoft Research
Carey Priebe
Johns Hopkins University

Article

Keywords:

Posted Date: January 20th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1244827/v1

Representation Ensembling for Synergistic Lifelong Learning with Quasilinear Complexity

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

Abstract. In biological 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 starts from a blank slate, or *tabula rasa*, using 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 catastrophic forgetting). Many recent approaches for continual or lifelong learning have attempted to *maintain* performance given new tasks. But striving to avoid forgetting sets the goal unnecessarily low: the goal of lifelong learning, whether biological or artificial, should be to improve performance on both past tasks (backward transfer) and future tasks (forward transfer) with any new data. Our key insight is that even though learners trained on other tasks often cannot make useful decisions on the current task (the two tasks may have non-overlapping classes, for example), they may have learned *representations* that are useful for this task. Thus, although ensembling decisions is not possible, ensembling representations can be beneficial whenever the distributions across tasks are sufficiently similar. Moreover, we can ensemble representations learned independently across tasks in quasilinear space and time. We therefore propose two algorithms: representation ensembles of (1) trees and (2) networks. Both algorithms demonstrate both forward and backward transfer in a variety of simulated and real data scenarios, including tabular, image, and spoken, and adversarial tasks. This is in stark contrast to the reference algorithms we compared to, all 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 biological learning, learning is 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 [2]. In classical machine learning, the system often starts with essentially zero knowledge, a "tabula rasa", and is optimized for a single task [3, 4]. While it is relatively easy to *simultaneously* optimize for multiple tasks (multi-task learning) [5], it has proven much more difficult to *sequentially* optimize for multiple tasks [6, 7]. 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 [8, 9]. This is in contrast to many biological learning settings, such as the second language learning setting mentioned above.

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 [10–14]. 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.

Approaches from both camps demonstrate some degree of continual (or lifelong) learning [18]. In particular, they can sometimes learn new tasks while not catastrophically forgetting old tasks. However, as we will show, many state of the art lifelong learning algorithms are unable to transfer knowledge forward, and none are able to transfer knowledge backward with small sample sizes where it is particularly

[1]Johns Hopkins University (JHU), [2]Baylor College of Medicine, [3]University of Cambridge [4]Microsoft Research,
[†] denotes equal contribution, [*] corresponding author: jdey4@jhu.edu

important. This inability to synergistically learn has been identified as one of the key obstacles limiting the capabilities of artificial intelligence [19, 20].

Our work builds on the ideas introduced in Progressive Neural Networks (ProgNN) [16], in which new tasks yield additional representational capacity. However, although ProgNN's are able to transfer forward, they fail to transfer backward. Moreover, as we will show, ProgNN requires quadratic space and time complexity in sample size. Our key innovation is the introduction of ensembling independent representations, rather than ensembling decisions (as in random forests [21] or network ensembles [22]). This is in contrast to ensembling representations that are conditionally dependent on the past representations (like gradient boosting trees [23] and ProgNN). By virtue of learning them independently, we reduce computational time and space from quadratic to quasilinear (i.e., linear up to polylog terms).

We implement two complementary synergistic learning algorithms, one based on decision forests (Syngeristic Forests, SYNF), and another based on deep networks (Synergistic Networks, SYNN). Both SYNF and SYNN demonstrate forward and backward transfer, while maintaining computational efficiency. Simulations illustrate their learning capabilities, including performance properties in the presence of adversarial tasks. We then demonstrate their learning capabilities in vision and language benchmark applications. Although the algorithms presented here are primarily resource building, we illustrate that they can effectively leverage prior representations. This ability implies that the algorithm can convert from a "juvenile" resource building state to the "adult" resource recruiting state – all while maintaining key synergistic learning capabilities and efficiencies.

2 Background

2.1 Classical Machine Learning Classical supervised learning [24] 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 $P_{X,Y} \in \mathcal{P}_{X,Y}$ 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 $P_{X,Y}$ [24]. A learning algorithm is evaluated on its generalization error (or expected risk): $\mathbb{E}\left[R(f(\mathbf{S}_n))\right]$, where the expectation is taken with respect to the true but unknown distribution governing the data, $P_{X,Y}$. The goal is to choose a learner f that learns a hypothesis h that has a small generalization error for the given task [25].

2.2 Lifelong Learning Lifelong learning generalizes classical machine learning in a few ways: (i) instead of one task, there is an environment $\mathcal{T}$ of (possibly infinitely) many tasks, (ii) data arrive sequentially, rather than in batch mode, and (iii) there are computational complexity constraints on the learning algorithm and hypotheses. This third requirement is crucial, though often implicit. Consider, for example, 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.

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, because the previously experienced tasks may recur, so we are concerned about maintaining and improving performance on those tasks as well. Previous work in lifelong learning falls loosely into two algorithmic camps: (i) continually updating a fixed parametric model as new tasks arrive, and (ii) adding resources as new tasks arrive. Some approaches additionally store or replay previously encountered data to reduce forgetting [26–28]. For our purposes, whether the algorithm stores new parameters, or new data, is irrelevant; the question is merely the capacity increased *at all*. So the question of merit is how much capacity changes with new data and/or tasks, not whether that increase is due to storing more parameters or more data. 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 task agnostic [29]) 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 nine 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 below. 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) [30]. 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.

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, 31]; for implementation details, see Appendix D.

3 Evaluation Criteria Others have previously introduced criteria to evaluate transfer, including forward and backward transfer [32, 33]. These definitions typically compare the difference, rather than the ratio, between learning with and without transfer. Pearl [19] introduced the transfer benefit ratio, which builds directly off relative efficiency from classical statistics [25]. Our definitions are closely related to his. *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. Let R^t be the risk associated with task t, and $\mathbf{S}_n^t$ be the data from $\mathbf{S}_n$ that is specifically associated with task t, so $R^t(f(\mathbf{S}_n^t))$ is the risk on task t of the hypothesis learned by f only on task t data, and $R^t(f(\mathbf{S}_n))$ denotes the risk on task t of the hypothesis learned on all the data.

Definition 1 (Learning Efficiency). *The learning efficiency of algorithm f for given task t with sample size n is $\mathsf{LE}_n^t(f) := \mathbb{E}\left[R^t\left(f(\mathbf{S}_n^t)\right)\right] / \mathbb{E}\left[R^t\left(f(\mathbf{S}_n)\right)\right]$. We say that algorithm f has learned task t with data $\mathbf{S}_n$ if and only if $\mathsf{LE}_n^t(f) > 1$.*

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 risk 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. Formally, let $N^t = \max\{i : T_i = t\}$, be the index of the last occurrence of task t in the data sequence. Let $\mathbf{S}_n^{\leq t} = \{(X_1, Y_1, T_1), ..., (X_{N^t}, Y_{N^t}, T_{N^t})\}$ be all data up to and including that data point.

Definition 2 (Forward Learning Efficiency). *The forward learning efficiency of f for task t given n samples is $\mathsf{FLE}_n^t(f) := \mathbb{E}\left[R^t\left(f(\mathbf{S}_n^t)\right)\right] / \mathbb{E}\left[R^t\left(f(\mathbf{S}_n^{\leq t})\right)\right]$.*

We say an algorithm (positive) 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.

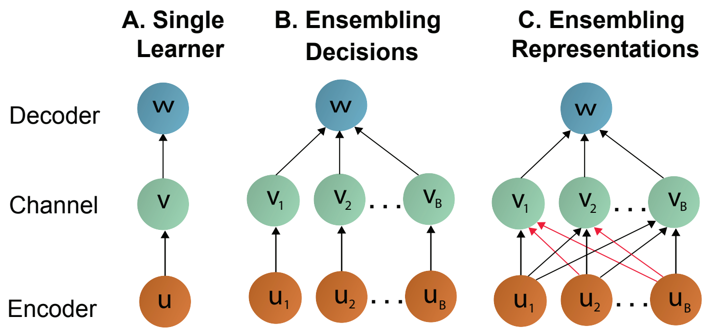

Figure 1: Schemas of composable hypotheses. Ensembling decisions (as output by the channels) is a well-established practice, including random forests and gradient boosted trees. Ensembling representations (learned by the encoders) was previously used in lifelong learning scenarios, but were not trained independently, thereby enabling inference or forgetting.

One can also determine the rate of *backward* transfer by comparing $R^t\left(f(\mathbf{S}_n^{\leq t})\right)$ to the risk of the hypothesis learned having seen the entire training dataset. More formally, backward learning efficiency is the expected ratio of the risk 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* $\text{BLE}_n^t(f) := \mathbb{E}\left[R^t\left(f(\mathbf{S}_n^{\leq t})\right)\right] / \mathbb{E}\left[R^t\left(f(\mathbf{S}_n)\right)\right]$.

We say an algorithm (positive) backward learns task t if and only if $\text{BLE}_n^t(f) > 1$. In other words, if $\text{BLE}_n^t(f) > 1$, then the algorithm has used data associated with future tasks to improve performance on previous tasks.

After observing m tasks, the extent to which the LE for the j^{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:

$$\text{LE}_n^t(f) = \frac{\mathbb{E}\left[R^t\left(f(\mathbf{S}_n^t)\right)\right]}{\mathbb{E}\left[R^t\left(f(\mathbf{S}_n)\right)\right]} = \frac{\mathbb{E}\left[R^t\left(f(\mathbf{S}_n^t)\right)\right]}{\mathbb{E}\left[R^t\left(f(\mathbf{S}_n^{\leq t})\right)\right]} \times \frac{\mathbb{E}\left[R^t\left(f(\mathbf{S}_n^{\leq t})\right)\right]}{\mathbb{E}\left[R^t\left(f(\mathbf{S}_n)\right)\right]}.$$

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 $\mathcal{T}$, we say an agent has **synergistically learned** if the agent has positively learned for each task t, i.e., $\log \text{LE}_n^t(f) > 0$ for all $t \in \mathcal{T}$. In contrast, we say an agent has **catastrophically forgotten**, if it has negatively learned for all the tasks.

4 Representation Ensembling Algorithms

Our approach to lifelong learning is based on an information theoretic hypothesis decomposition of an encoder, channel, and decoder [34, 35] (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 $\tilde{\mathcal{X}}$ [36, 37]. The channel $v : \tilde{\mathcal{X}} \mapsto \Delta_{\mathcal{Y}}$ maps the transformed data into a posterior distribution (or, more generally, a score) on the response space $\mathcal{Y}$. Finally, a decoder $w : \Delta_{\mathcal{Y}} \mapsto \mathcal{Y}$, produces a predicted label. See Appendix A for a 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 [38] and bagging [39]—indeed, decision forests are a canonical example of a decision function operating on a collection of B outputs [21]. 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 that single tree's guess as to the class of any probability 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. However, the encoders learned independently across different tasks may have learned useful **representations** that the tasks can mutually leverage. Thus, a further generalization of the decomposition in Figure 1B allows for each channel to ensemble the encoders (Figure 1C). 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 ensembles 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. See Appendix B for a concrete example. We developed two different representation ensembling algorithms.

The key to both of our algorithms is the realization that both forests and networks partition feature space into a union of polytopes [40]. Thus, the internal representation learned by each can be considered a sparse vector encoding which polytope a given sample resides in.

In either of the cases, as new data from a new task arrives, our algorithm first builds a new independent encoder (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, thereby enabling forward transfer. If new data arrive from an old task, it can leverage 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. Note that while updating the previous task channels with the cross-task posteriors, we do not need to subsample the previous task data (see Appendix C for implementation details and pseudocodes).

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 [21, 41]. The leaf nodes of each decision tree partition the input space $\mathcal{X}$ [42]. The representation of $x \in \mathcal{X}$ corresponding to a single tree can be a one-hot encoded L_b-dimensional vector with a 1 in the location corresponding to the leaf x falls into of tree b. The representation of x resulting from the collection of trees simply concatenates the B one-hot vectors from the B trees. Thus, the encoder u_t is the mapping from $\mathcal{X}$ to a B-sparse vector of length $\sum_{b=1}^{B} L_b$. The channel then learns the class-conditional posteriors by populating the cells of the partitions and taking class votes with out-of-bag samples, as in "honest trees" [42–44]. Each channel outputs the average normalized class votes across the collection of trees, adjusted for finite sample bias [45]. The decoder w_t averages the posterior estimates and outputs the argmax to produce a single prediction. Recall that honest decision forests are universally consistent classifiers and regressors [44], 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" [45]. 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 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. In practice, we use the architecture described in van de Ven et al. [28] as "5 convolutional layers followed by 2 fully-connected layers each containing 2,000 nodes with ReLU non-linearities and a softmax output layer." We trained this network using cross-entropy loss and the Adam optimizer [46] to learn the encoder. The channels are learned via k-Nearest Neighbors (k-NN) [47]. 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 [47]. We use $k = 16 \log_2 n$, which satisfies these conditions. The decoder is the same as above.

SynN was motivated by ProgNN, but 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. This difference 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 ProgNN and 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 (which we refer to as 'fixed resources' hereafter). We therefore quantify the computational space and time complexity of the internal representation of a number of algorithms, using both theoretical analysis and empirical investigations. We also study the representation capacity of these algorithms. We use the soft-O notation $\tilde{\mathcal{O}}$ to quantify complexity [48]. 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. Table 1 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 1: 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
semiparametric	T	T^2	n^2	nT	n^2	ProgNN
semiparametric	T	T	n	n	n	DF-CNN
semiparametric	T	$T + n$	n	n	n	SynN
nonparametric	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. $\mathrm{S{\small YN}N}$ eliminates the lateral connections between columns of the network, thereby reducing space complexity down to $\tilde{\mathcal{O}}(T)$. $\mathrm{S{\small YN}N}$ stores all the data to enable backward transfer, but retains linear time complexity.

$\mathrm{S{\small YN}F}$ 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 Illustrating Synergistic Learning with $\mathrm{S{\small YN}F}$

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. We sample a total 750 samples from XOR, followed by another 750 samples from XNOR.

$\mathrm{S{\small YN}F}$ 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, $\mathrm{S{\small YN}F}$ continues to improve on XOR given XNOR data, demonstrating backward transfer. Now consider the generalization error on *XNOR* (Figure 2Bii). Both $\mathrm{S{\small YN}F}$ 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 $\mathrm{S{\small YN}F}$ 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). $\mathrm{S{\small YN}F}$ 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.

Synergistic learning in adversarial environments Statistics has a rich history of *robust learning* [49], and machine learning has recently focused on *adversarial learning* [50]. 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

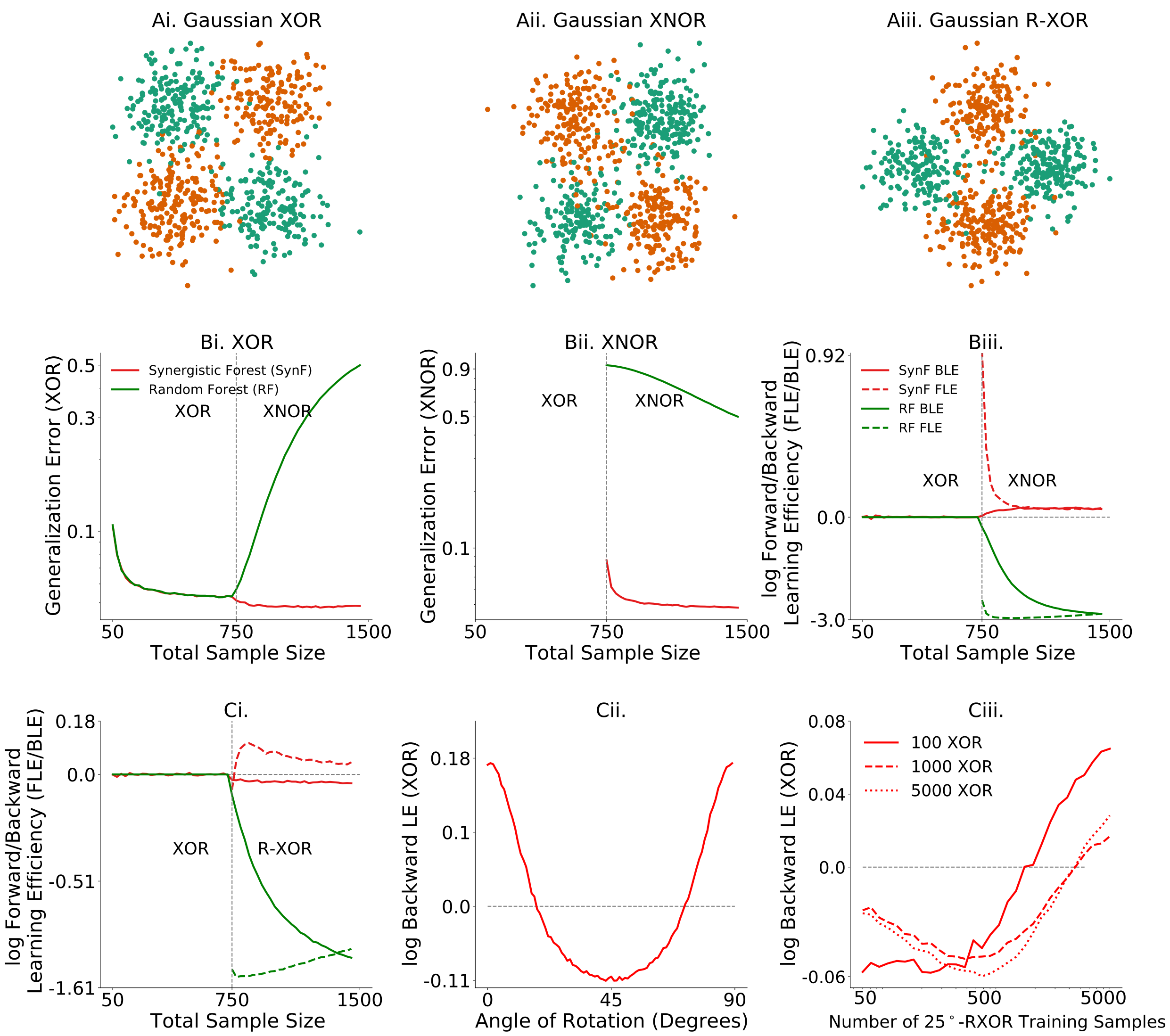

Figure 2: **Synergistic Forests demonstrate forward and backward transfer.** (*A*) 750 samples from: (*Ai*) Gaussian XOR, (*Aii*) XNOR, which has the same optimal discriminant boundary as XOR, and (*Aiii*) R-XOR, which has a discriminant boundary that is uninformative, and therefore adversarial, to XOR. (*Bi*) Generalization error for XOR, and (*Bii*) XNOR of both SynF (red) and RF (green). SynF outperforms RF on XOR when XNOR data is available, and on XNOR when XOR data are available. (*Biii*) Forward and backward learning efficiency of SynF are positive for all sample sizes, and are negative for all sample sizes for RF. (*Ci*) In an adversarial task setting (XOR followed by R-XOR), SynF gracefully forgets XOR while positively forward transferring to R-XOR, whereas RF demonstrates 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 increases at different rates for different XOR (target task) sample numbers with respect to sample size for 25°-XOR (source task).

XOR rotated by 45° (R-XOR) (Figure 2Aiii). Training on R-XOR therefore impedes the performance of SynF on XOR, and thus backward transfer falls below one, demonstrating graceful forgetting [51] (Figure 2Ci). Because R-XOR is more difficult than XOR for SynF (because the discriminant boundaries are oblique [52]), 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.

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). 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 and the target 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.

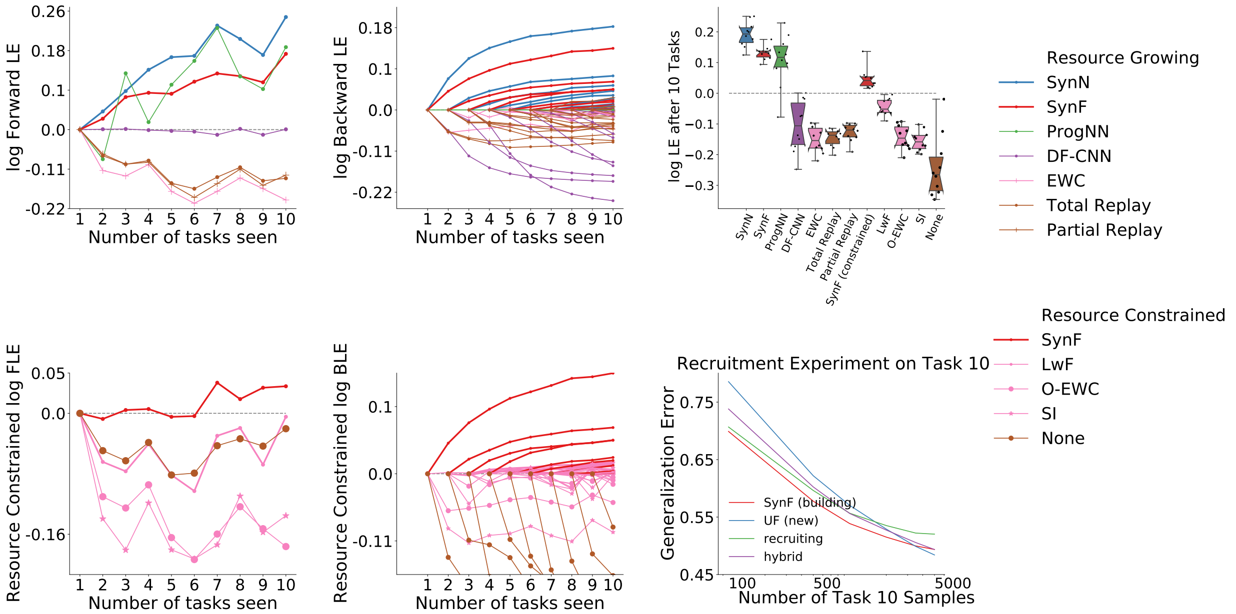

Figure 3: **Performance of different algorithms on the CIFAR 10x10 vision experiments.** *Top left and center*: Forward and backward transfer efficiency for various resource building algorithms. SYNF and SYNN consistently demonstrate both forward and backward transfer for each task, whereas ProgNN and DF-CNN do not. *Bottom left and center*: 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.

5.3 Real data experiments We consider two modalities for real data experiments: vision and language. Below we provide a detailed analysis of the performance of lifelong learning algorithms in vision data; Appendix F provides details for our language experiments, which have qualitatively similar results illustrating that SYNF and SYNN| are modality agnostic, sample and computationally efficient, lifelong learning algorithms.

The CIFAR 100 challenge [53], 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. 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**?**], and it improves on a particular task by leveraging knowledge from the other tasks. Therefore, for our following experiments, we are particularly interested in the behavior of our representation ensembling algorithms in the low training sample size regime. The below experiments use only 500 training samples per task. For the corresponding experiments using higher training samples per task (5,000 samples), see Appendix Figure 4.

Resource Growing Experiments We first compare SynF and SynN to state-of-the-art resource growing algorithms: ProgNN and DF-CNN (Figure 3, top panels). Both SynF and SynN demonstrate positive forward transfer for every task (SynF increases nearly monotonically), indicating they are robust to distributional shift in ways that ProgNN and DF-CNN are not. SynN and SynF uniquely 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, while neither ProgNN nor DF-CNN exhibit catastrophic forgetting, they also do not exhibit any positive backward transfer. Final transfer efficiency per task is the transfer efficiency associated with that task having seen all the data. SynF and SynN both demonstrate positive final transfer efficiency for all tasks (synergistic learning), whereas ProgNN and DF-CNN both exhibit negative final transfer 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 3, 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 3, top right). Interestingly, when using 5,000 samples per task, replay methods are able to demonstrate positive forward and backward transfer (Supplementary Figure 4), 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 3, top right panel) whereas all other algorithms except ProgNN suffer 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 3, 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 4A 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 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 4, 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*.

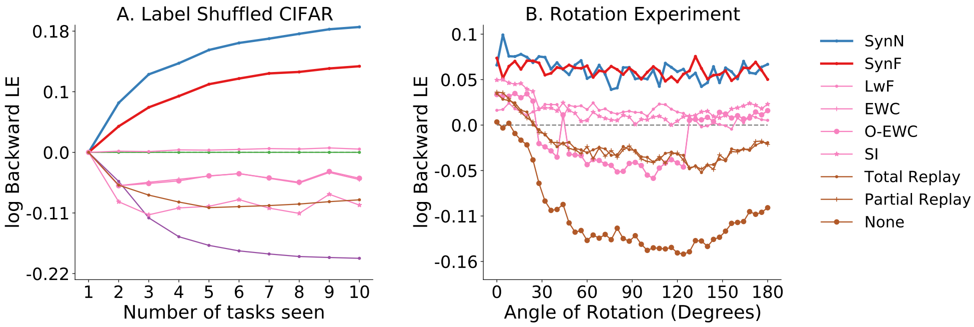

Figure 4: **Extended CIFAR 10x10 experiments.** (*A*) Shuffling class labels within tasks two through nine with 500 samples each demonstrates both SYNF and SYNN can still achieve positive backward transfer, and that the other algorithms still fail to transfer. (*B*) SYNF and SYNN are nearly invariant to rotations, whereas other approaches are more sensitive to rotation.

6 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. 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 [54].

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 computational 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 would obviate the need to store all the data by using a generative model.

While we employed quasilinear representation ensembling to address catastrophic forgetting, the paradigm of ensembling *representations* rather than *learners* 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 [55] and genomics [56]. Similarly, federated learning is becoming increasingly central in artificial intelligence, due to its importance in differential privacy [57]. 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 [58].

Finally, our quasilinear representation ensembling approach closely resembles the constructivist view of brain development [59, 60]. 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 [61]. 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] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.

[9] 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.

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

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

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

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

[22] Nicolas Pinto, David Doukhan, James J DiCarlo, and David D Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. PLoS Comput. Biol., 5(11):e1000579, November 2009.

[23] 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.

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

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

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

[27] 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.

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

[30] 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.

[31] 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.

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

[33] 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.

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

[35] 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.

[36] 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.

[37] 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.

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

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

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

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

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

[43] 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.

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

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

[46] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

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

[48] 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.

[49] Peter J Huber. Robust statistical procedures, volume 68. Siam, 1996.

[50] 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.

[51] 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.

[52] 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.

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

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

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

[56] 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.

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

[58] 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.

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

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

[61] 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.

[62] 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 $[L] = \{1, 2, \ldots, L\}$ 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 voter is simply a matter of counting the fraction of observations in each leaf per class. So, the voter 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: $\{v_l := \mathbb{P}[y_i = 1|\tilde{x}_i = l], \forall l \in [L]\}$ and $v(\tilde{x}) = v_{\tilde{x}}$. 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 voter 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 $\tilde{x}^{(1)}_{n_1+1}, \ldots, \tilde{x}^{(1)}_n$. 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 $\tilde{x}^{(2)}_{n_1+1}, \ldots, \tilde{x}^{(2)}_n$. Third, we train a new channel, v_2. To do so, v_2 is trained on the outputs from both encoders, that is, $\{(\tilde{x}^{(j)}_i, y_i)\}_{i=n_1+1}^n$ 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, 62].

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 $\tilde{x}^{(1)}_1, \ldots, \tilde{x}^{(1)}_{n_1}$. 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: $\tilde{x}^{(2)}_1, \ldots, \tilde{x}^{(2)}_{n_1}$. 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. We propose 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 representer 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 ▷ a representer set
(2) $\mathcal{I}_{OOB}^t$ ▷ a set of the indices of OOB data
function SynX.fit($t, (\mathbf{x}^t, \mathbf{y}^t)$)
  $u_t, \mathcal{I}_{OOB}^t \leftarrow$ X.fit($\mathbf{x}^t$, $\mathbf{y}^t$) ▷ train a representer X on bootstrapped data
  return $u_t, \mathcal{I}_{OOB}^t$
end function

Algorithm 2 Add a new SynX voter for the current task.

Input:
(1) t ▷ current task number
(2) $\boldsymbol{u}_t = \{u_t\}_{t'=1}^t$ ▷ the set of representers
(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 ($t' \neq t$) voters for task t
function SynX.add_voter($t, \boldsymbol{u}_t, (\mathbf{x}_t, \mathbf{y}_t), \mathcal{I}_{OOB}^t$)
  $v_{tt} \leftarrow u_t$.add_voter($(\mathbf{x}_t, \mathbf{y}_t), \mathcal{I}_{OOB}^t$) ▷ add the in-task voter using OOB data
  for $t' = 1, \ldots, t-1$ **do** ▷ update the cross task voters for task t
    $v_{tt'} \leftarrow u_{t'}$.add_voter($\mathbf{x}_t, \mathbf{y}_t$)
  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 $\textsc{SynX}$ voter for the previous tasks.

Input:
(1) t ▷ current task number
(2) u_t ▷ representer 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_VOTER(t, u_t, D)
    for t' = 1, ..., t − 1 do                         ▷ update the cross task voters
        v_{t't} ← u_t.get_voter(x_{t'}, y_{t'})
    end for
    return v
end function
```

Algorithm 4 Predicting a class label using $\textsc{SynX}$.

Input:
(1) $x \in \mathbb{R}^p$ ▷ test datum
(2) t ▷ task identity associated with x
(3) $\boldsymbol{u}$ ▷ all T reperesenters
(4) $\boldsymbol{v}_t$ ▷ voter for task t
Output: $\hat{y}$ ▷ a predicted class label

```
function ŷ = SynX.PREDICT(t, x, v_t)
    T ← SynX.get_task_number()                        ▷ get the total number of tasks
    p̂_t = 0                                           ▷ p̂_t is a K-dimensional posterior vector
    for t' = 1, ..., T do                             ▷ update the posteriors calculated from T task voters
        p̂_t ← p̂_t + v_{tt'}.predict_proba(u_{t'}(x))
    end for
    p̂_t ← p̂_t / T
    ŷ = argmax_i(p̂_t)        ▷ find the index i of the elements in the vector p̂_t with maximum probability
    return ŷ
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 [45]) and a $\textsc{SynF}$. 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

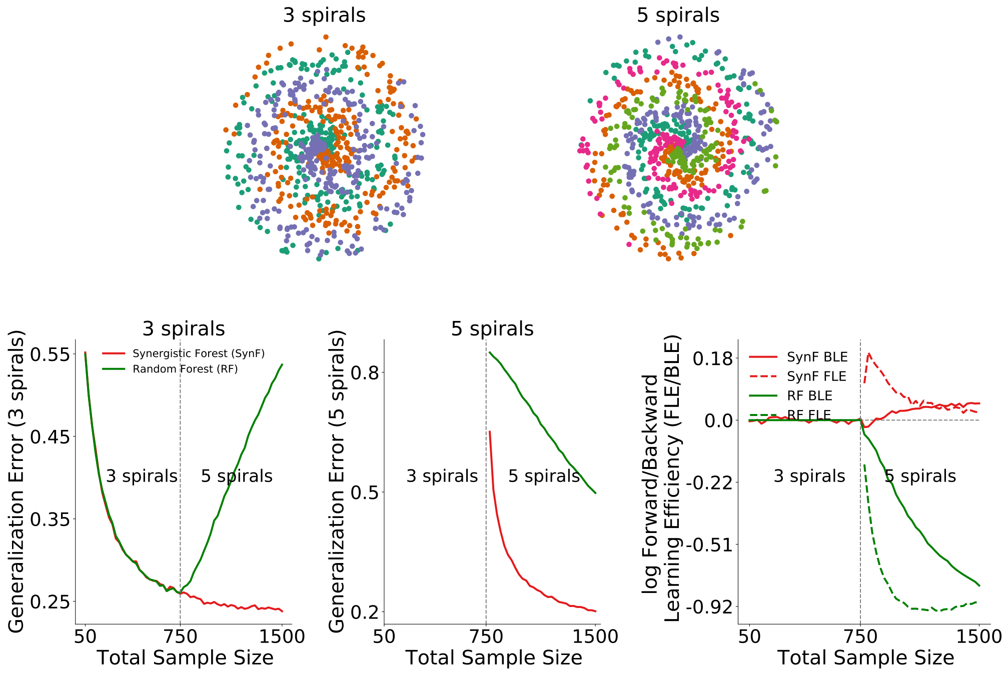

Figure 1: *Top*: 750 samples from 3 spirals (left) and 5 spirals (right). *Bottom left*: SynF outperforms RF on 3 spirals when 5 spirals data is available, demonstrating *backward* transfer in SynF. *Bottom center*: SynF outperforms RF on 5 spirals when 3 spirals data is available, demonstrating *forward*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

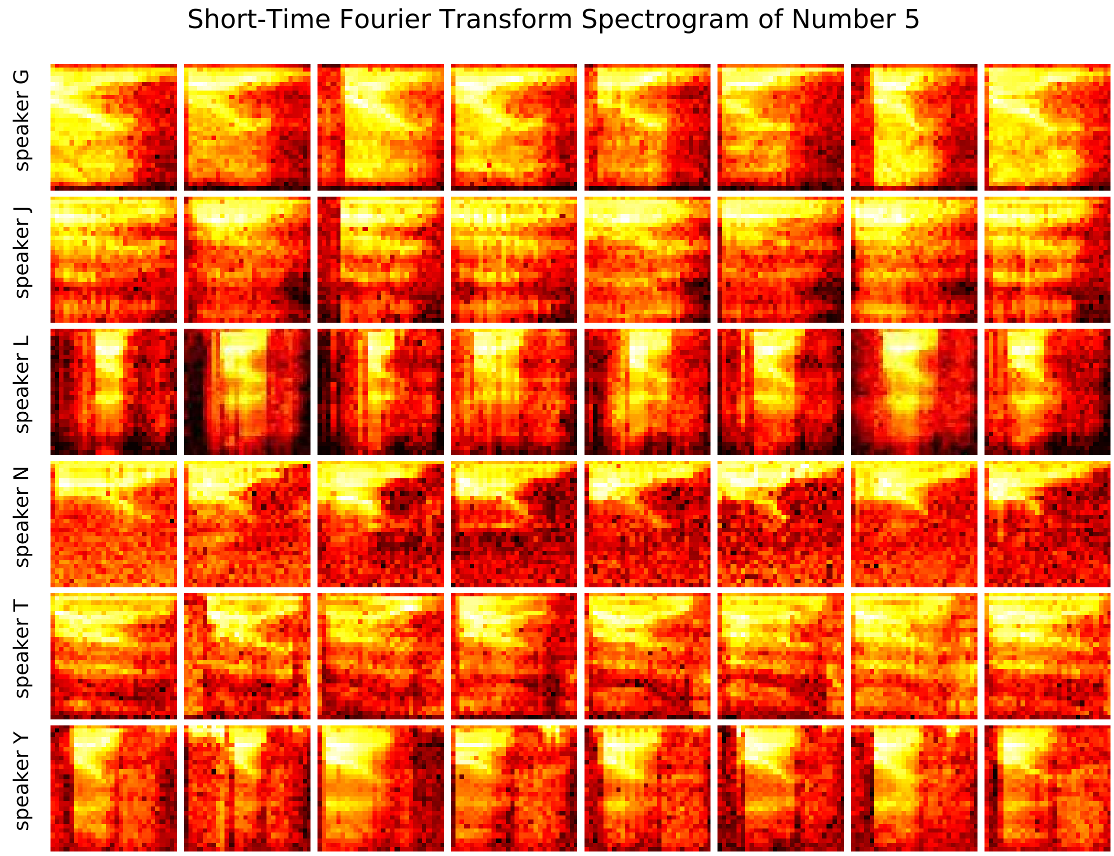

Figure 2: Spectrogram extracted from 8 different recordings of 6 speakers uttering the digit '5'.

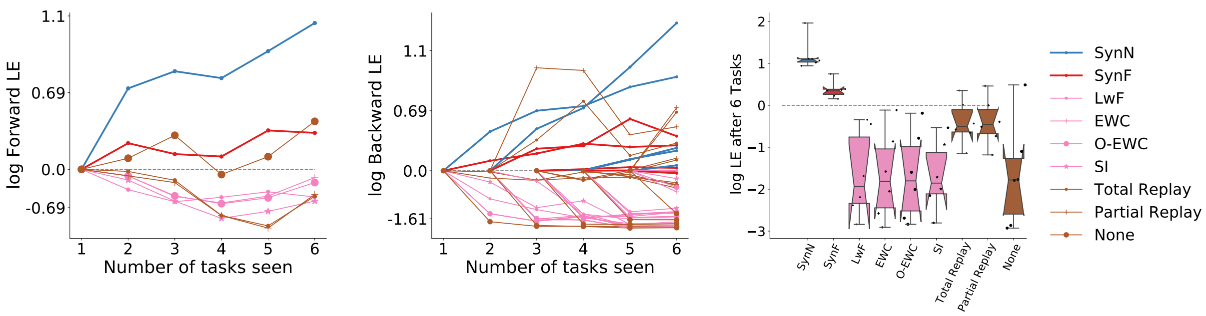

Figure 3: Both SYNF and SYNN show positive forward and backward transfer as well as synergistic learning for the spoken digit tasks, in contrast to other methods, some of which show only forward transfer, others show only backward transfer, with none showing both, and some showing neither.

recording, a spectrogram was extracted using Hanning windows of duration 16 ms with an overlap of 4 ms between the adjacent windows. The spectrograms were resized down to 28×28. The extracted spectrograms from 8 random recordings of '5' for 6 speakers are shown in Figure 2. For each Monte Carlo repetition of the experiment, spectrograms extracted for each task were randomly divided into 55% train and 45% test set. As shown in Figure 3, 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 forkward transfer, others show only backward transfer, with none showing both, and some showing

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

neither.

F.2 CIFAR 10x10 Supplementary Table 3 shows the image classes associated with each task number. Supplementary Figure 4 is the same as Figure 3 but with 5,000 training samples per task, rather than 500. Notably, with 5,000 samples, replay methods are able to transfer both forward and backward as well. However, note that although total replay outperforms both SynF and SynN with large sample sizes, it is not a *bona fide* lifelong learning algorithm, because it requires n^2 time. Moreover, the replay methods will eventually forget as more tasks are introduced because it will run out of capacity.

F.3 CIFAR Label Shuffling Supplementary Figure 5 shows the same result as the label shuffling from Figure 4, 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.4 CIFAR 10x10 Repeated Classes We also considered the setting where each task is defined by a random sampling of 10 out of 100 classes with replacement. This environment is designed to demonstrate the effect of tasks with shared subtasks, which is a common property of real world lifelong learning tasks. Supplementary Figure 6 shows transfer efficiency of SynF and SynN on Task 1.

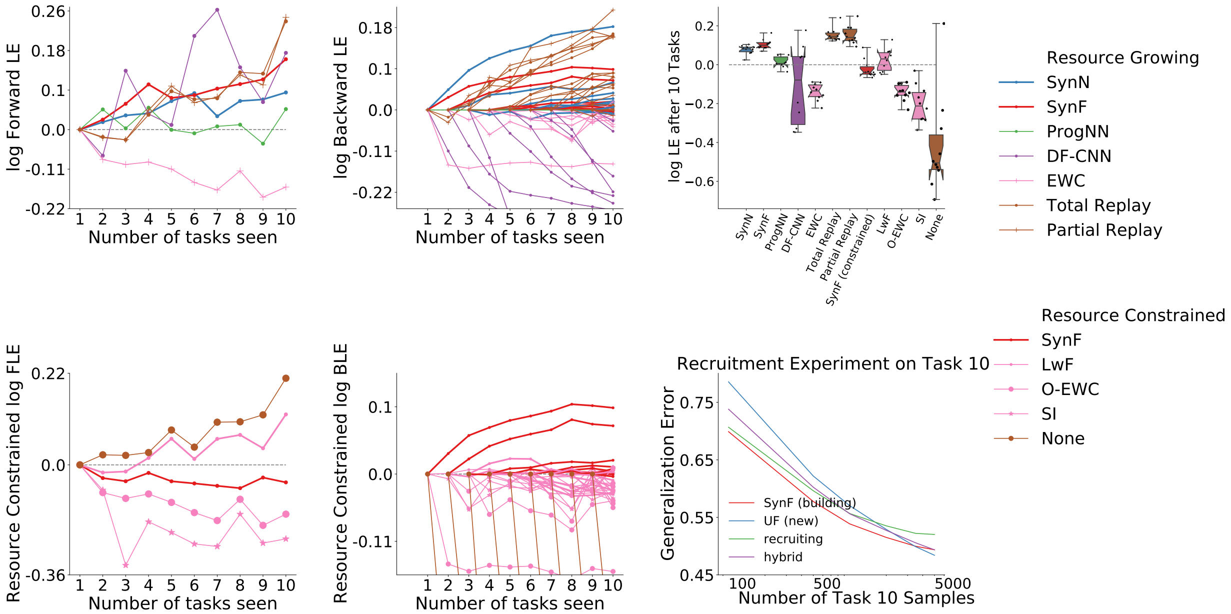

Figure 4: Performance of different algorithms on CIFAR 10x10 vision dataset for $5{,}000$ training samples per task. SynN maintains approximately the same forward transfer (top left and bottom left) and backward transfer (top center and bottom center) efficiency as those for 500 samples per task whereas 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.

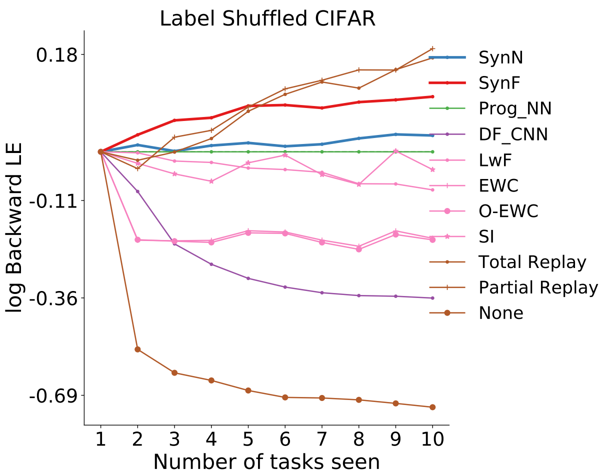

Figure 5: Label shuffle experiment on CIFAR 10x10 vision dataset for $5{,}000$ training samples per task. Shuffling class labels within tasks two through nine with 5000 samples each demonstrates both SynF and SynN can still achieve positive backward transfer, and that the other algorithms that do not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

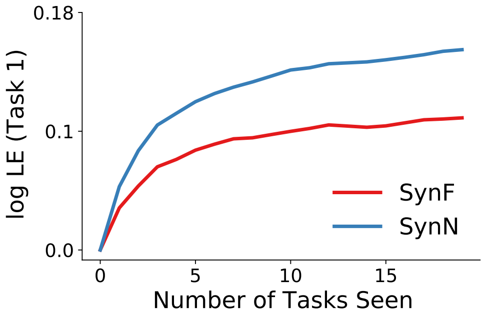

Figure 6: SynF and SynN transfer knowledge effectively when tasks share common classes. Each task is a random selection of 10 out of the 100 CIFAR-100 classes. Both SynF and SynN demonstrate monotonically increasing transfer efficiency for up to 20 tasks.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- JDeyCSflat.pdf
- JDeyEPCflat.pdf