The 37th Canadian Conference on Artificial Intelligence
DOI: 0

Liquid Ensemble Selection for Continual Learning

Carter Blair† Ben Armstrong† Kate Larson†
† University of Waterloo

Abstract

Continual learning aims to enable machine learning models to acquire new knowledge from a shifting data distribution without forgetting what has already been learned. Such shifting distributions can be broken into disjoint subsets of related examples called contexts; training each ensemble member on a different context makes it possible for the ensemble as a whole to achieve much higher accuracy with less forgetting than a naive model. We address the problem of selecting which models within an ensemble should learn on any given data and which should predict. By drawing on work from delegative voting we develop an algorithm for using delegation to dynamically select which models in an ensemble are active. We explore various delegation methods and performance metrics, ultimately finding that delegation can provide a significant performance boost over naive learning in the face of distribution shifts.

Keywords: Continual Learning, Liquid Democracy, Delegation, Social Choice

1. Introduction

Machine learning models often operate under the assumption that the data they are trained on and the data on which they will make predictions are independently and identically distributed (i.i.d.). However, this assumption does not reflect the reality of the world we live in. In practical settings, data distributions are constantly shifting. Deployed machine learning models that operate as though the world is static will eventually become highly inaccurate or will begin to enforce outdated knowledge about the world – often resulting in harmful bias. Instead, models must adapt to the world as it changes.

Continual learning is a machine learning framework where models learn incrementally on training data that arrives in an ordered stream from a shifting distribution. Naively learning on such data allows models to remain performant on recent data but leads to a problem known as *catastrophic forgetting* wherein previously learned information is forgotten [1, 2]. Continual learning usually focuses on *learning* from a non-stationary stream without forgetting; however, we may also want to make good predictions on this stream. As such, ensembles, which are collections of models whose predictions are aggregated, are a natural choice since we can select which members are predicting and which are learning independently.

Ensembles have long been used in many machine-learning settings. In standard learning situations, ensembles typically outperform single classifiers and demonstrate a strong ability to leverage diverse models for a common purpose. Conceptually, ensembles are well-suited to both learning and predicting on shifting data: By using different classifiers to learn on different underlying distributions, the ensemble can dynamically select classifiers to rely upon when making predictions and avoid forcing a single model to forget previous knowledge in order to stay up-to-date. Existing work has already found that ensembles can be applied to a continual learning setting [3–5].

We argue that the recent use of delegative voting for ensemble pruning lays a foundation for a new continual learning algorithm. This delegative voting framework, known as *liquid democracy*, allows ensemble members to participate directly in training and inference or to transitively delegate their action to another classifier. While many continual learning

algorithms rely upon explicit knowledge about changes in the distribution [3, 5], or remember useful training examples so they may be re-learned [6–8], our procedure relies only upon knowledge about the recent learning performance of each classifier. Using this information, our algorithm makes delegation decisions which further two goals: (1) selecting the classifiers most well suited to learn or predict on the current distribution and (2) weighting the chosen classifiers appropriately in order to optimize their prediction accuracy.

Our algorithm is highly generic; we explore a variety of existing delegation methods from liquid democracy and several measures of classifier performance. We can consider any single-valued measure such as accuracy, precision, recall or f1-score. In experiments across multiple types of continual learning, we demonstrate the utility of adapting ideas from liquid democracy into continual learning. In particular, our contributions are as follows:

- A novel framework for continual learning based on principles of liquid democracy that requires minimal knowledge of the learning setting.
- New definitions of classifier performance in terms of learning rate and recent performance, tailored to the specific demands of continual learning.
- Empirical validation of our approach via experiments on classic continual learning benchmarks, demonstrating a strong ability to enhance ensemble performance.

Through these contributions, our work establishes a new direction for research in continual learning, highlighting the potential of integrating concepts from delegative voting into machine learning to create more adaptable and diverse ensembles.

2. Related Work

2.1. Ensembles in Social Choice

Ensembles rely on aggregation methods to combine outputs from many classifiers into a single prediction. A great deal of work spanning multiple decades has applied social choice knowledge to ensemble learning. This has been illustrated through theoretical connections between ensemble learning and axiomatic results in social choice [9] and by experimental evaluation of many common voting rules for classifier prediction aggregation [10, 11].

Only one line of work has considered the application of liquid democracy to ensembles. In a setting with exactly one round of delegation, liquid democracy has been shown to not provide any significant benefit over traditional ensembles [12]. However, delegation can also be viewed as a form of ensemble pruning and reweighting. Using iterative delegations in an incremental learning setting (without the distribution shift common in continual learning), Armstrong and Larson showed that liquid democracy typically achieves higher accuracy than a full ensemble while also dramatically reducing training costs [13]. On some datasets, their method attained higher accuracy than the well-known ensemble algorithm Adaboost [14]. However, this result was highly data-dependant.

2.2. Ensembles in Continual Learning

Ensembles are a useful method to improve performance in continual learning [3–5]. On one end of the spectrum, some ensemble methods initialize a new member for each new context they encounter [5]. However, these methods have high resource usage and require context labels in training, limiting their applicability. Other methods, such as SEED [3] and CoSCL [4], use a fixed number of models in the ensemble to avoid a linear increase in parameters with the number of contexts. These methods also use the context label during training.

Beyond ensembles, replay methods, which store examples from each context, are often used in continual learning [6–8]. However, the storage of examples introduces extra memory requirements and can be a privacy concern. To avoid these issues, some methods use a

generative model to generate samples for replay [15–17], but these methods introduce extra technical complexity and may not perform well in data-sparse settings.

3. Model

Our model bridges two fields of study: machine learning and social choice. We consider an epistemic social choice setting where voters are instantiated as an ensemble of machine learning classifiers. Voters (classifiers) may vote for one of several alternatives in pursuit of a single "correct" alternative; this corresponds to the classification action of a classifier. We focus on a setting where data has not only features and labels but also a *context* associated with some distribution. Within this setting, we explore algorithms for dynamically re-weighting voters/classifiers using delegative techniques. That is, classifier weights are controlled by "delegation" between classifiers, and only non-delegating classifiers take part in inference or learning. Classifiers may participate actively (they learn/predict on examples directly) or may delegate to another classifier, thereby increasing that classifier's prediction weight and not learning or predicting.

Specifically, we consider an ensemble of classifiers $V = \{v_1, v_2, ..., v_n\}$, which we equivalently refer to as voters, and a non-stationary data stream $\mathcal{D}$ containing m classes. Data arrives in batches of B samples over T steps, with each batch associated with some context $c_t \in \mathcal{C}$; $\mathcal{D} = \{(\mathcal{X}_1, \mathcal{Y}_1, c_1), ..., (\mathcal{X}_\mathcal{T}, \mathcal{Y}_\mathcal{T}, c_T)\}$. Each example in a dataset can be decomposed into three pieces: features $\mathcal{X}$, a class label $\mathcal{Y}$, and a context label $\mathcal{C}$. Together the class label and context label make up a *global label* $\mathcal{G}$. Depending upon the setting, this stream may include training data, testing data, or training data *and* testing data. Figure 1 gives an example of the difference between a class and context labels.

A delegation function, $d : V \times T \to V$, determines which classifiers are learning or doing inference on any given batch. A delegation function value $d(v_i, t) = v_j$ indicates that, during batch t, v_i delegates to v_j. In most cases, the batch is clear from context, and we write $d(v_i) = v_j$ to indicate a delegation in the current batch. By default, classifiers delegate to themselves. Classifiers that delegate to anyone other than themselves are referred to as "delegators" and are *inactive* while a classifier that delegates to itself is *active* and may be referred to as a "guru." The set of all gurus is found through repeated application of the delegation function until a self-delegation is reached: $G_t(V) = \{v \in V | d^*(v, t) = v\}$. Only classifiers that are gurus participate in learning or inference, depending on the setting.

The weight of each classifier is controlled by delegation. Specifically, a weight vector W_t contains the weight associated with each classifier on batch t. If no classifiers delegate (that is, they all delegate to themselves), then each classifier has a weight of 1. Generally, each classifier has a weight equal to the number of delegations they receive, including their own:

$$W_t[i] = \begin{cases} 0 & \text{if } v_i \notin G_t(V) \\ |\{v_j \in V | d^*(v_j, t) = v_i\}| & \text{otherwise} \end{cases}$$

An estimate of the performance of each classifier is continually refined. Performance can be measured as prediction accuracy or any other single-valued metric that reflects the performance of a classifier on a batch of data. $\mathrm{A} = [a_{t,i}]$ stores the performance of each classifier v_i on each batch t. We are often interested in the trend of a classifier's performance as it learns. That is, over recent batches, has it become more or less accurate? We use this to approximate two measures: (1) whether or not the classifier is still improving its performance/benefits from continued training, and (2) as a proxy for whether a context shift has happened. If a classifier's performance drops rapidly, this is an indication that the underlying data has shifted distributions. To evaluate this performance trend, we calculate the slope of the linear regression line of the performance values of each classifier over some parameter w of recent batches. We denote the performance slope of v_i as $q_{i,w}$ and all

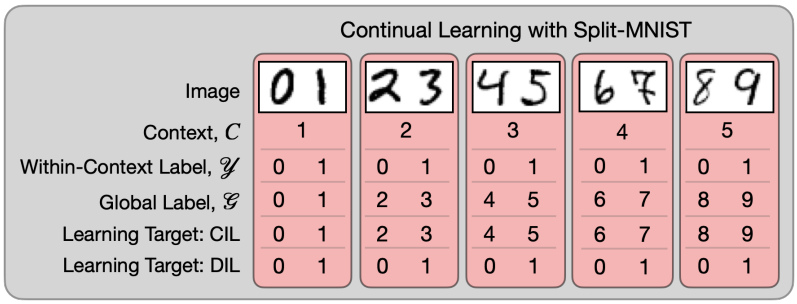

Figure 1. Label types and learning targets within continual learning. Classes are grouped into disjoint contexts which are concatenated to form a data stream. Class Incremental Learning (CIL) learns the global label – a unique combination of context label and within-context label. Domain Incremental Learning (DIL) learns only the within-context label. (This figure is inspired by a similar figure from Ven, Tuytelaars, and Tolias [2].)

such slopes as Q. The set of all classifiers with a higher slope than v_i is $N^+(v_i) = \{v_j \in V | q_{j,w} > q_{i,w}\}$. Intuitively, this corresponds to the classifiers that are "better" than v_i; those showing more recent improvement in accuracy. Typically, we refer to the current weight and estimated performance of classifier v_i as w_i and q_i, respectively, only including a time subscript where necessary.

To generate a prediction, each guru in the ensemble generates class probabilities for each possible class. These are given a weight equal to the guru's weight and summed to get a weighted probability for each class. The class with the highest weighted probability is the ensemble's prediction.

3.1. **Continual Learning**

In continual learning, a learner attempts to learn from a non-stationary stream of data arriving in batches, each associated with some context. A context may appear in multiple contiguous batches, but, importantly, after a context stops arriving, it does not return during the training phase. This can lead to what is commonly referred to as "catastrophic forgetting," which is characterized by the learner forgetting how to classify samples from contexts learned early in training after further learning in other contexts has occurred.

We explore two of the standard scenarios within continual learning, differentiated based upon the information available to the learner and the label space [2, 18]:

(1) In *class-incremental learning*, contexts have disjoint class labels. Classifiers only know the current context during training, not testing. The classifier must learn to predict both the label and context given only the input. Formally, the goal is to learn a mapping $f : \mathcal{X} \to \mathcal{Y} \times \mathcal{C}$.

(2) In *domain-incremental learning*, each context has the same class labels, but they may appear at different frequencies in different contexts. The goal is to learn a mapping $f : \mathcal{X} \to \mathcal{Y}$.

Figure 1 gives an example of the learning target of both learning methods. Algorithm 1 outlines the general behaviour of our continual learning training algorithms: Voters all start as gurus then, for each batch of data: The ensemble receives the batch, has all gurus learn the batch, and updates delegations based on training performance. Note that in all settings, we assume the training data is ordered or non-stationary, but there are no assumptions about the test set. We can construct the test set as useful for evaluation. For example, it is common to report test accuracy over all contexts and within context. We may assume, in some cases, that the test set is non-stationary.

Algorithm 1 Continually Learning Liquid Democracy Ensemble

Input: Batch size B, number of classifiers n, delegation mechanism $DelMech$
Initialize all classifiers $\{v_i\}_{i=1}^{n}$ and all delegations to $d(v_i) = v_i$
for t = 1, ..., T **do**
  Receive batch $\{(x_j, y_j)\}_{j=1}^{B}$ from data stream
  for $v_i \in V$ **do**
    v_i makes predictions $\{\hat{y}_{i,j}\}_{j=1}^{B}$ on current batch
    if $d(v_i) = v_i$ (classifier is guru) **then**
      v_i learns on batch
    end if
  end for
  Update delegations: $DelMech(V)$
end for

4. Delegation Mechanisms

We develop a delegation mechanism suitable for continual learning that aims to be adaptive to context shifts while incorporating existing delegation methods as a parameter. Our mechanism, k-Best-Accuracy-Trend (k-BAT), does not depend on knowledge about the current context or store previous examples. Rather, the mechanism uses a weak signal about the recent performance of each classifier in the ensemble to determine whether/where each classifier should delegate.

4.1. k-Best-Accuracy-Trend Delegations

Our algorithm is presented in Algorithm 2 and here in words. In continual learning, data is organized into contexts which are further subdivided into batches. k-BAT works over batches, using two parameters to record performance and make decisions: A `metric` parameter evaluates the performance of classifiers and a `delegation probability function` determines where classifiers will delegate.

During learning, k-BAT is called once per batch after the active voters in the ensemble have learned on the batch. When it begins, k-BAT calculates and records the prediction performance of all classifiers on the recent batch according to the given `metric` (this can be any signal about performance such as accuracy, f1-score, etc.). Subsequently, for each classifier, linear regression is performed over the most recent w batches and the slope of the result is recorded (a positive slope indicates that a classifier is generally improving, while a negative slope indicates the reverse).

After recording the performance, delegation begins. The k classifiers with the best performance trend are chosen to act as gurus. Each remaining classifier chooses a single other classifier to whom they delegate. Each delegation is made according to the given `delegation probability function`, which may use accuracy, weight, or other information to determine the probability that each classifier will delegate to each other classifier.

Note that k-BAT only operates during *training*; during test time, all classifiers in the ensemble are active. The idea is that when making predictions, classifiers trained on a specific context are more confident and accurate when predicting classes from that context, while those not trained on that context are less confident and spread their probability mass more evenly between the classes. Since we make final predictions by summing the class probabilities across ensemble members, this means the accurate, confident classifiers mainly determine the final prediction on classes from the context they were trained on.

In contrast to other methods, we focus on keeping the architecture simple and general. We do not use a generative model or replay buffer or leverage context labels in training or

Algorithm 2 Best Accuracy Trend Delegation Mechanism

Input: V, w, `metric`, `delegation_probability_function`
$P \leftarrow n \times n$ matrix initialized to 0
for $v_i \in V$ **do**
  $q_i \leftarrow$ linear regression slope of `metric` over last w batches.
end for
for $v_i \in V$ **do**
  for each $v_j \in V$ **where** $i \neq j$ **do**
    if $q_i > q_j$ and $d(v_i) = v_j$ **then**
      $d(v_i) := v_i$
    else
      $P_{i,j} :=$ `delegation_probability_function`(v_i, v_j)
    end if
  end for
  Select a v_j according to probabilities $P_{i,\_}$
  $d(v_i) := v_j$
end for

testing, and we use a fixed number of models in our ensemble to limit parameter growth. Not relying on context labels in training allows our method to work in a more diverse range of settings, including both class-incremental and domain-incremental learning settings.

Below, we describe the individual delegation probability functions that we explore.

4.2. Delegation Probability Functions

Definition 1. *The **Random Better** Delegation Probability Function assigns delegation probabilities such that each classifier with a higher regression slope than the delegator is selected with equal probability.*

$$p^{rand\_better}(v_i, v_j) = \begin{cases} \frac{1}{|N^+(v_i)|} & j \in N^+(v_i) \\ 0 & otherwise \end{cases}$$

Definition 2. *The **Proportional Better** Delegation Probability Function assigns delegation probabilities such that each classifier with a higher regression slope than the delegator is assigned a probability proportional to their regression slope value. Shown here are delegation probabilities before normalization.*

$$p^{prop\_better}(v_i, v_j) \propto \begin{cases} \frac{q_j - q_i}{\sum_{v_k \in N^+(v_i)} q_k - q_i} & j \in N^+(v_i) \\ 0 & otherwise \end{cases}$$

Definition 3. *The **Proportional Weighted** Delegation Probability Function assigns delegation probabilities such that each classifier with a higher regression slope than the delegator is assigned a probability proportional to both their regression slope value and the weight of the delegate's guru. A lower weight leads to a higher delegation probability. Shown here are delegation probabilities before normalization.*

$$p^{prop\_weighted}(v_i, v_j) \propto \begin{cases} \frac{1}{w_{d^*(j)}} \frac{q_j - q_i}{\sum_{v_k \in N^+(v_i)} q_k - q_i} & j \in N^+(v_i) \\ 0 & otherwise \end{cases}$$

4.3. **Student-Expert Delegations**

The Student-Expert extends delegation to test time by running delegation separately on two sets of classifiers: one set which learns (students) and another which is used for inference (experts). Typically, in continual learning, evaluation is done after training on each context individually or with the assumption that the test stream is drawn i.i.d from all contexts. This can be useful for assessing the degree of forgetting, but in reality, the test stream is likely to be non-stationary, similar to the training stream. We can leverage the non-stationary nature of the test stream with a method that dynamically shifts which members of the ensemble are making predictions.

The Student-Expert delegation mechanism does just this by adding delegation to the test phase. Delegation dynamically shifts which members of the ensemble are making predictions. The Student-Expert mechanism is an extension of the k-Best-Accuracy-Trend (k-BAT) delegation mechanism described in subsection 4.1. It maintains k_s "student" classifiers and k_e "expert" classifiers. $k - BAT$ is run normally for each batch of data to pick k_s students who learn on that batch. Separately, we select k_e experts, which will make predictions on the batch. Currently, we use only the Proportional Better delegation probability function to select student gurus. Students measure performance using a parameterized window size of w and a given performance metric as in k-BAT, while experts measure performance by comparing classifier prediction accuracy on only the most recent batch.

5. **Experiments**

We evaluate our algorithms along several axes – number of gurus, performance metric, and delegation mechanism. To do this, we run three experiments: First, a simple experiment demonstrates the behaviour of an ensemble using k-BAT within a class-incremental setting. Second, an in-depth experiment explores parameterizations of our algorithms for class-incremental learning. Finally, a similar experiment applies the same algorithms to domain-incremental learning. Our results show strong performance across both settings and surprising differences in optimal parameters for class- and domain-incremental learning.

5.1. **Experimental Setup**

Data: We use two MNIST datasets common throughout continual learning research: Split MNIST and Rotated MNIST [2]. Split MNIST consists of grey-scale images of digits from 0 to 9, all scaled to the same dimensions [19]. Feature values are pixels, and the class is the digit value. We use Split MNIST as a stream of data partitioned into 5 contexts, each consisting of exactly two random digits. We use Split MNIST in the class-incremental setting where we learn the global label, i.e. the digit value (0-9).

In the domain-incremental setting, we use a slightly more complex data set: Rotated MNIST. This data again consists of grey-scale images of digits, as in Split MNIST. However, the digits are now rotated around the center of the image from anywhere between 180° and −180°. We initialize Rotated MNIST into a stream consisting of 5 disjoint contexts. Each context corresponds to a specific rotation amount and contains only images corresponding to that rotation. The context label is the rotation amount, and the within-context label (the domain-incremental learning target) is the digit value (0-9). Each context's rotation is sampled uniformly at random.

Parameter Values: We explore a range of parameters in both learning settings. We evaluate k-BAT with three performance metrics: accuracy, balanced accuracy and F1 score; as well as $k \in \{1, 2, 3, 4\}$. We explore each delegation probability function and report on the best one. Additionally, we use between 1 and 4 experts in Student-Expert delegation.

Experiments use ensembles of size 8 except the larger experiment on class-incremental learning, which explored an ensemble of size 30 with up to 11 gurus, and the smaller illustrative experiment, which used an ensemble size of 5.

Our models are all simple feed-forward neural networks with two hidden layers which use the Adam optimizer, implemented using Pytorch [20]. In line with previous work that compares ensemble methods with multiple learners to methods which only use a single learner, we keep the total number of trainable parameters for each continual learning method roughly equal [4]. That is, when comparing the performance of an ensemble of 8 classifiers to a single network, each network in the ensemble has roughly one-eighth the number of trainable parameters as the solitary network. In the class-incremental setting, k-BAT and the Student-Expert method use a window size, w, of 50 batches. In the domain-incremental setting, these methods use a window size of 400. In all cases, we use a batch size of 128.

Benchmarks: We compare our algorithms with a wide range of existing continual learning methods implemented in the Avalanche Continual Learning library [21]. These methods can largely be divided upon two axes: memory and context knowledge. Our methods, k-BAT and Student Expert, do not store previous examples or receive context labels while learning. Memory-based methods (GEM [22], MIR [23], Replay [24]) store previously seen data to avoid forgetting it but do not necessarily use task identities. Methods that do not rely on memory often focus on slowing down updates to certain parts of a model (e.g. neural network weights useful for learning about one class are not updated). Some such methods are explicitly given task identities (MAS [25], CWR [26]) while others are not (LFL [27], RWalk [28]).

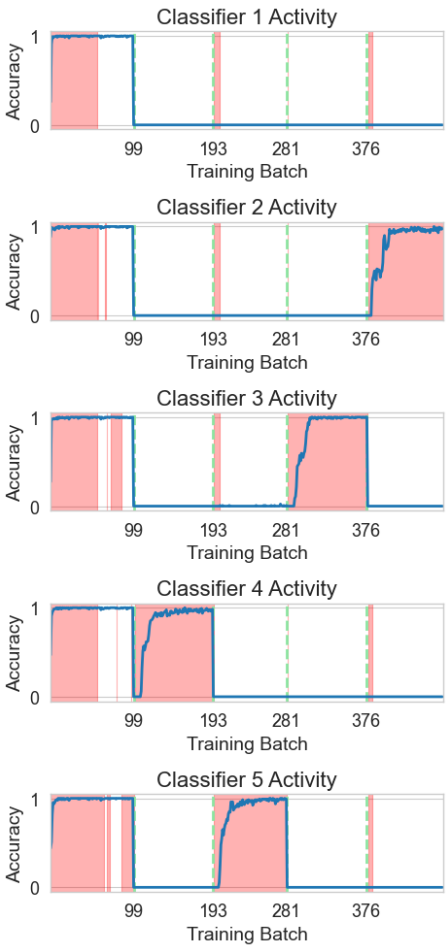

Figure 2. Active learning periods of each classifier on Split MNIST using k-BAT with $k = 1$ and probabilistic better delegation. Red periods indicate a classifier is actively learning. With a window size of 50, all classifiers learn on the first 50 batches. Subsequently, delegation begins, and one classifier learns at a time. At context shifts (green lines) various classifiers briefly begin to learn but k-BAT effectively picks a single classifier to learn the majority of a context without any knowledge of the context label.

5.2. **Results**

In our first experiment, we construct a simplistic scenario using the Split MNIST data, which is meant to illustrate the behaviour of k-BAT. We plot each classifier's learning curve and the periods in which they were active on the training set. Figure 2 shows that the classifiers divide and conquer, with one classifier learning the majority of each context after the first. This is notable as k-BAT does not use the context label and does not explicitly aim to recognize when the context changes.

At the beginning of learning, all classifiers remain active until they learn on w batches and delegation begins. Classifiers that have learned on fewer batches have more plasticity

and are likely to learn more quickly, leading to a higher regression slope. Thus, classifiers not trained on a previous context learn more quickly than classifiers already trained on a context. When the context switches, the trained classifier's performance drops rapidly, leading to a negative regression slope. Then, the classifier previously learning will delegate to a classifier that learns quickly on the new context.

5.2.1. **Class-Incremental Learning**

Our second experiment investigates the k-BAT and Student-Expert methods using varying parameter values in our delegative framework. In particular, we vary the number of active learning gurus during training for both the k-BAT and the Student-Expert method, and we vary the number of active predicting gurus for the Student-Expert method. We also vary the delegation probability function and the classifier competency metric for k-BAT.

Table 1 shows the average accuracy of each method on the full test set as well as the average accuracy in each context on the Split MNIST dataset. Both k-BAT and the Student-Expert method significantly outperform naive methods (Full Ensemble and Single Learner). The Student-Expert method is the only one to leverage order in the test set and thus significantly outperforms all similar methods that do not rely on memory. Order in the test set is rarely available, so we separate these results from others. However, even k-BAT, which does not receive any batch labels during testing, is more accurate than other methods that do not rely on memory.

The performance on each of the five contexts reveals that k-BAT and Student-Expert successfully remember information from the first four contexts, indicating a strong ability to avoid catastrophic forgetting. In regard to methods which predict on the whole test set statically, our k-BAT method with 1 active learning guru performs particularly well, achieving greater than zero accuracy in each context.

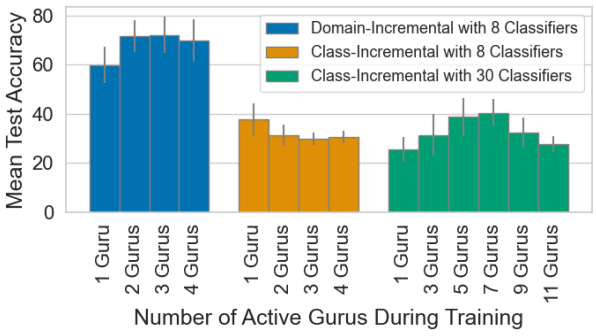

Figure 3. Test accuracy for k-BAT on domain-incremental learning and two sizes of ensemble doing class-incremental learning. Results are averaged over the probability delegation functions and delegation metrics. While the small-scale experiment on class-incremental learning performs best with only 1 guru, other settings are optimized with larger numbers of gurus.

The Student-Expert model shows the benefit that comes with structure in an online test stream. Overall, we see that Student-Expert with 1 guru during both training and testing achieved a 91.75% test accuracy while k-BAT with 1 guru achieved only a 43.27% accuracy. However, even k-BAT is much higher than all other methods that similarly don't rely on memory.

Figure 3 shows that, in the larger class-incremental setting with 30 classifiers, having as many as 7 gurus can benefit performance. However, in smaller ensembles, 1 guru appears optimal. We found no significant differences in accuracy when varying the delegation probability function and classifier competency metric in k-BAT.

5.2.2. **Domain-Incremental Learning**

In our third experiment, we investigate the performance of k-BAT and Student-Expert in the domain-incremental setting. The results of this experiment can be seen in Table 2. The Student-Expert method significantly outperforms all other methods that do not use memory,

Method	Mean Acc.	C_1	C_2	C_3	C_4	C_5
Student-Expert	91.75 ± 0.39	93.05 ± 0.85	89.95 ± 1.07	92.08 ± 0.42	92.90 ± 0.15	90.79 ± 0.33
GEM	**95.78 ± 0.22**	98.39 ± 0.23	**95.19 ± 0.65**	**94.98 ± 0.76**	**95.79 ± 0.35**	94.55 ± 0.33
Replay	72.95 ± 1.76	80.85 ± 3.00	60.96 ± 4.77	51.45 ± 6.30	74.20 ± 4.73	**97.27 ± 0.35**
MIR	52.42 ± 10.00	60.00 ± 21.01	57.34 ± 14.61	64.48 ± 22.05	0.12 ± 0.21	80.16 ± 14.99
k-BAT	43.27 ± 3.38	73.18 ± 14.16	2.45 ± 5.10	93.28 ± 4.34	35.88 ± 17.49	11.54 ± 17.10
CWR	19.98 ± 0.01	**99.88 ± 0.05**	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
LwF	19.37 ± 0.14	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	96.87 ± 0.70
MAS	25.26 ± 3.18	18.04 ± 11.09	1.99 ± 2.87	0.34 ± 0.57	9.48 ± 8.93	96.47 ± 0.26
RWalk	19.43 ± 0.06	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	97.17 ± 0.28
Single Learner	19.44 ± 0.05	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	97.19 ± 0.26
Full Ensemble	19.28 ± 0.08	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	96.40 ± 0.41

Table 1. Test accuracy on Split MNIST for all methods averaged over 10 trials in the class-incremental learning setting evaluated after training on all contexts. Results are grouped into 3 sections; the top section shows the performance possible without memory and when leveraging structure within the test set. The middle section compares against pre-existing methods that rely on memory. The bottom section compares methods that do not use memory. Our methods show that ensembles are able to provide significant performance boosts when leveraging structure in test data (Student-Expert with 1 train, 1 test guru), as well as when not memorizing previous examples or being given the context label (k-BAT with $k = 1$, Probabilistic Better delegation probability function and balanced accuracy as the performance metric).

Method	Mean Acc.	C_1	C_2	C_3	C_4	C_5
Student Expert	92.46 ± 1.39	93.59 ± 0.62	89.21 ± 3.27	92.34 ± 2.41	92.12 ± 3.94	95.05 ± 0.68
GEM	**95.41 ± 0.60**	**94.66 ± 1.44**	**94.93 ± 1.50**	**94.72 ± 0.90**	**95.71 ± 0.81**	97.05 ± 0.21
Replay	75.25 ± 8.73	64.86 ± 16.41	71.25 ± 21.33	64.59 ± 16.54	78.40 ± 11.79	97.13 ± 0.36
MIR	61.97 ± 11.91	44.83 ± 24.31	57.07 ± 32.44	48.97 ± 23.19	61.83 ± 22.91	97.15 ± 0.31
k-BAT	74.78 ± 7.24	73.92 ± 19.46	64.51 ± 25.03	65.59 ± 18.22	85.08 ± 9.95	84.82 ± 8.49
CWR	55.00 ± 8.06	65.32 ± 16.09	43.12 ± 19.88	43.83 ± 14.78	62.34 ± 17.54	60.40 ± 13.44
LwF	70.81 ± 9.06	56.92 ± 21.97	62.23 ± 30.19	59.89 ± 20.91	77.81 ± 14.57	**97.20 ± 0.27**
MAS	65.78 ± 7.57	62.66 ± 19.10	58.79 ± 21.73	51.29 ± 17.00	72.80 ± 13.26	83.38 ± 4.10
RWalk	67.77 ± 9.77	53.48 ± 21.63	62.51 ± 28.99	54.77 ± 21.32	71.48 ± 16.10	96.61 ± 0.39
Single Learner	62.29 ± 11.36	45.83 ± 24.37	57.09 ± 32.70	48.97 ± 22.85	62.51 ± 21.16	97.04 ± 0.18
Full Ensemble	59.42 ± 11.86	42.10 ± 25.15	54.10 ± 33.59	45.58 ± 22.74	59.09 ± 23.05	96.23 ± 0.44

Table 2. Test accuracy for all methods averaged over 10 trials in the domain-incremental learning setting using the Rotated MNIST dataset. All methods are evaluated after training on all contexts. k-BAT uses the random better delegation probability function and F1 score as the performance metric and 2 gurus during training. Student-Expert uses the probabilistic better delegation probability function and accuracy as the metric with 2 classifiers active during training and testing. Most methods show strong test performance on the most recently learned data while exhibiting moderate forgetfulness on previously learned data. k-BAT exhibits much stronger performance than existing methods that do not rely on replay. The Student-Expert algorithm (top row) highlights the performance possible on an ordered test stream, even without memory.

achieving an average test accuracy of 92.46% with 2 active gurus during training and testing. This is primarily due to the fact that Student-Expert operates on an ordered, online test stream, as discussed previously. Of the remaining methods that don't use memory, k-BAT with $k = 2$ achieves the highest average accuracy of 74.78%.

Impressively, both Student-Expert and k-BAT perform similarly well in the first context as in some later contexts. Overall, both of our methods avoid significant forgetting.

When averaging over all parameter configurations for k-BAT, using $k = 2$ or $k = 3$ resulted in the best performance. Similar to our findings in the class-incremental setting, each performance metric and delegation probability function leads to similar performance.

6. Discussion

In this paper, we have developed new ensemble training algorithms based on delegative techniques for two types of continual learning: class-incremental and domain-incremental. Our first algorithm, k-BAT, approximately selects a unique ensemble member to learn each new context without any knowledge about the context itself. k-BAT effectively teaches different members of the ensemble on different distributions and drastically reduces the catastrophic forgetting common to continual learning settings. Our Student-Expert delegation mechanism extends k-BAT to allow delegation during testing and improves performance in the presence of ordered test data. This avoids catastrophic forgetting and showcases the power of delegation for dynamically selecting classifiers to both learn and make predictions.

Curiously, in our class-incremental learning experiments with only 8 classifiers in the ensemble, we found that 1 guru led to the highest accuracy. This is not the case with domain-incremental learning, where multiple gurus are superior. Intuitively, when there are more classifiers than contexts, having multiple classifiers learn on a context should have better performance than when just one classifier is learning. Our result suggests that either (1) individual classifiers are quite strong in the class-incremental setting and that more gurus may be useful when the learning task is more difficult, or (2) there is room for improvement in the delegation algorithm. In either case, further investigation will be useful.

Generally, our experiments have shown the power of ensembles for continual learning and have demonstrated the ability to learn effectively across context shifts without explicit knowledge of the context. However, a great deal of work could extend this research in compelling directions. Some specific paths include:

- Optimal delegations: We have no guarantee of the quality of our delegative process. Both theoretical and further experimental work will be important to identify an upper bound on the performance improvement granted by our delegation mechanisms.
- New performance metrics for delegation: Our delegation mechanisms use a generic signal of quality to decide delegations. We experimented with common metrics such as accuracy and f1-score; however, other metrics may lead to better outcomes.
- Further comparison with existing methods to avoid catastrophic forgetting. We compared our results with three methods well suited for domain-incremental learning, however these methods prove to work poorly on the class-incremental setting.

References

[1] R. M. French. "Catastrophic Forgetting in Connectionist Networks". In: *Trends in cognitive sciences* 3.4 (1999), pp. 128–135.

[2] G. M. van de Ven, T. Tuytelaars, and A. S. Tolias. "Three Types of Incremental Learning". In: *Nature Machine Intelligence* 4.12 (2022), pp. 1185–1197.

[3] G. Rypeść, S. Cygert, V. Khan, T. Trzciński, B. Zieliński, and B. Twardowski. *Divide and not Forget: Ensemble of Selectively Trained Experts in Continual Learning*. arXiv:2401.10191 [cs]. Jan. 2024.

[4] L. Wang, X. Zhang, Q. Li, J. Zhu, and Y. Zhong. "CoSCL: Cooperation of Small Continual Learners is Stronger Than a Big One". en. In: *Computer Vision – ECCV 2022*. Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2022, pp. 254–271.

[5] R. Aljundi, P. Chakravarty, and T. Tuytelaars. "Expert Gate: Lifelong Learning With a Network of Experts". In: 2017, pp. 3366–3375.

[6] P. Pan, S. Swaroop, A. Immer, R. Eschenhagen, R. Turner, and M. E. E. Khan. "Continual Deep Learning by Functional Regularisation of Memorable Past". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 4453–4464.

[7] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny. "Efficient Lifelong Learning with A-GEM". In: *International Conference on Learning Representations*. 2019.

[8] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. "iCaRL: Incremental Classifier and Representation Learning". en. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, July 2017, pp. 5533–5542.

[9] D. M. Pennock, P. Maynard-Reid II, C. L. Giles, and E. Horvitz. "A Normative Examination of Ensemble Learning Algorithms." In: *ICML*. 2000, pp. 735–742.

[10] C. Cornelio, M. Donini, A. Loreggia, M. S. Pini, and F. Rossi. "Voting with Random Classifiers (VORACE): Theoretical and Experimental Analysis". In: *Autonomous Agents and Multi-Agent Systems* 35.2 (2021), p. 22.

[11] F. Leon, S.-A. Floria, and C. Bădică. "Evaluating the Effect of Voting Methods on Ensemble-Based Classification". In: *2017 IEEE International Conference on INnovations in Intelligent Systems and Applications*. IEEE. 2017, pp. 1–6.

[12] B. Armstrong and K. Larson. "On the Limited Applicability of Liquid Democracy". In: *3rd Games, Agents, and Incentives Workshop at AAMAS* (2021).

[13] B. Armstrong and K. Larson. "Liquid Democracy for Low-Cost Ensemble Pruning". In: *Autonomous Agents and Multi-Agent Systems* (2024).

[14] Y. Freund and R. E. Schapire. "A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting". In: *European Conference on Computational Learning Theory*. Springer. 1995, pp. 23–37.

[15] G. M. van de Ven, H. T. Siegelmann, and A. S. Tolias. "Brain-inspired Replay for Continual Learning with Artificial Neural Networks". en. In: *Nature Communications* 11.1 (Aug. 2020).

[16] G. M. Van De Ven, Z. Li, and A. S. Tolias. "Class-Incremental Learning with Generative Classifiers". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Nashville, TN, USA: IEEE, June 2021, pp. 3606–3615.

[17] H. Shin, J. K. Lee, J. Kim, and J. Kim. "Continual Learning with Deep Generative Replay". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[18] L. Wang, X. Zhang, H. Su, and J. Zhu. "A Comprehensive Survey of Continual Learning: Theory, Method and Application". In: *arXiv preprint arXiv:2302.00487* (2023).

[19] F. Zenke, B. Poole, and S. Ganguli. "Continual Learning Through Synaptic Intelligence". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3987–3995.

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. "Pytorch: An Imperative sStyle, High-performance Deep Learning Library". In: *Advances in Neural Information Processing Systems* 32 (2019).

[21] V. Lomonaco, L. Pellegrini, A. Cossu, A. Carta, G. Graffieti, T. L. Hayes, M. De Lange, M. Masana, J. Pomponi, G. M. Van de Ven, et al. "Avalanche: An End-to-End Library for Continual Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 3600–3610.

[22] D. Lopez-Paz and M. Ranzato. "Gradient Episodic Memory for Continual Learning". In: *Advances in neural information processing systems* 30 (2017).

[23] R. Aljundi, L. Caccia, E. Belilovsky, M. Caccia, M. Lin, L. Charlin, and T. Tuytelaars. "Online Continual Learning with Maximally Interfered Retrieval". In: *ArXiv* abs/1908.04742 (2019).

[24] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P Dokania, P Torr, and M Ranzato. "Continual Learning with Tiny Episodic Memories". In: *Workshop on Multi-Task and Lifelong Reinforcement Learning*. 2019.

[25] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars. "Memory Aware Synapses: Learning What (not) to Forget". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 139–154.

[26] V. Lomonaco and D. Maltoni. "Core50: A New Dataset and Benchmark for Continuous Object Recognition". In: *Conference on robot learning*. PMLR. 2017, pp. 17–26.

[27] Z. Li and D. Hoiem. "Learning Without Forgetting". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.12 (2017), pp. 2935–2947.

[28] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. Torr. "Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 532–547.