UNSUPERVISED CONTINUAL LEARNING AND SELF-TAUGHT ASSOCIATIVE MEMORY HIERARCHIES*

James Smith, Seth Baer, Zsolt Kira & Constantine Dovrolis
College of Computing, Georgia Institute of Technology, Atlanta, GA
{jamessealesmith, sbaer8, zkira, constantine}@gatech.edu

ABSTRACT

We first pose the Unsupervised Continual Learning (UCL) problem: learning salient representations from a non-stationary stream of unlabeled data in which the number of object classes varies with time. Given limited labeled data just before inference, those representations can also be associated with specific object types to perform classification. To solve the UCL problem, we propose an architecture that involves a single module, called Self-Taught Associative Memory (STAM), which loosely models the function of a cortical column in the mammalian brain. Hierarchies of STAM modules learn based on a combination of Hebbian learning, online clustering, detection of novel patterns, forgetting outliers, and top-down predictions. We illustrate the operation of STAMs in the context of learning hand-written digits in a continual manner with only 3-12 labeled examples per class. STAMs suggest a promising direction to solve the UCL problem without catastrophic forgetting.

1 INTRODUCTION.

Unsupervised Continual Learning (UCL) involves learning from a stream of unlabeled data in which the data distribution or number/type of object classes vary with time. UCL is motivated by recent advances in Continual Learning (CL) (Hsu *et al.*, 2018; Parisi *et al.*, 2018) but also differs in that it is completely unsupervised and there are no priors on the data stream. In UCL the data stream includes unlabeled instances of both previously learned classes and, occasionally, new classes. This setting mirrors the natural world where known object types keep re-appearing – if they do not, it makes sense to forget them. Many CL methods involve some sort of "replay" – we argue that observing instances of known classes (perhaps infrequently) is equivalent to replaying previous instances.

To evaluate whether a given architecture can solve the UCL problem, we partition the time axis in distinct *learning phases*. During each phase, the data stream includes unlabeled examples from a constant set of classes (unknown to the architecture). At the end of each phase, we evaluate the architecture with a simple classification task. To do so, we provide a limited number of labeled instances per class. This labeled dataset is *not* available during the learning phase and it is only used to associate the class-agnostic representations that the architecture has learned with the specific classes that are present in the labeled dataset. This is different than Semi-Supervised Learning (SSL) methods (Springenberg, 2015; Oliver *et al.*, 2018; Miyato *et al.*, 2018) because SSL requires both labeled and unlabeled data during the training process. We have found one SSL method compatible with the UCL problem, the latent-feature discriminate model (M1) (Kingma *et al.*, 2014), and we present a variation of that method in the experimental section.

To solve the UCL problem, we have developed a neuro-inspired architecture based on a model of cortical-columns, referred to as Self-Taught Associative Memory (STAM). The connection between STAMs and cortical models is described in the next section. The architecture is a layered hierarchy of STAM modules that involve forward, feedback, and lateral connections. The hierarchy learns salient features through online clustering. Each feature is a cluster centroid. STAMs at different layers of the hierarchy learn centroids at different spatial resolutions (different receptive field sizes).

*Supported by the Lifelong Learning Machines (L2M) program of DARPA/MTO: Cooperative Agreement HR0011-18-2-0019

STAMs learn in an online manner through mechanisms that include novelty detection, forgetting outlier patterns, intrinsic dimensionality reduction, and top-down predictions.

STAMs have some superficial similarities with Convolutional Neural Networks (CNN) (Krizhevsky *et al.*, 2012) in that they are both layered and have increasing receptive field sizes. However, the STAM architecture learns in a Hebbian manner without the task-specific optimization requirement of CNNs. Further, the features learned by STAMs are highly interpretable (they are basically common patterns at different spatial resolutions), and they adapt to non-stationarities in the data distribution.

The STAM architecture has similarities with several unsupervised methods in machine learning. First, there is a large body of work in clustering-based methods for unsupervised learning – see Caron *et al.* (2018) for a recent review. Previous clustering-based methods, however, do not focus on intrinsic dimensionality, do not include mechanisms for detecting novel patterns and forgetting outliers, and do not leverage top-down feedback. In general, we do not consider iterative algorithms, such as the "deep clustering" architecture (Caron *et al.*, 2018), to be comparable with STAM in the UCL context because they require repetitive training epochs through the same data.

Machine learning methods such as Helmholtz machines Dayan *et al.* (1995) and Deep Predictive Coding Lotter *et al.* (2016) are similar to the STAM architecture in terms of how they use feedback connections - but they are not based on clustering and they have not been developed in the context of continual learning tasks, meaning that they assume a mostly stationary environment. Another approach to model the function of cortical columns has been pursued by Numenta George and Hawkins (2009). Even though our high-level position is the same (namely, that the basic building block of artificial neural networks should be a functional model of the cortical column, rather than individual neurons), the Numenta architecture ("Hierarchical Temporal Memory") is fundamentally different than the STAM architecture.

2 CORTICAL COLUMNS AND STAMS

The design of STAMs is based on the following points about cortical columns:

1) The cerebral cortex consists of the same six-layer module, referred to as cortical column, repeated throughout the cortex with minor anatomical differences Kaschube *et al.* (2010); Kaas (2012); Reid (2012); Miller (2016). The "canonical cortical circuit" by Douglas and Martin captures what is currently known about this module at the level of connections between the six cortical layers Douglas and Martin (2004); Douglas *et al.* (1989).

2) If the same cortical module is used in brain regions associated with very different function (e.g., the columns of V1 "see" visual features, the columns of A1 "hear" sounds, the columns of the prefontal cortex make plans), we can hypothesize that the cortical column circuit performs a very general but powerful computational function.

3) The structure of cortical columns is such that it can be viewed as a module with two input channels: a feedforward input channel from lower brain regions (such as the thalamus) or from lower-level cortical regions (e.g., V1 columns projecting to V2 columns), and a feedback input channel from higher-level cortical regions (e.g., from V2 columns to V1 columns). Symmetrically, a cortical column has two output channels: the feedback outputs towards lower-level cortical regions and other parts of the brain, and the feedforward outputs towards higher cortical regions. These feedforward/feedback channels are used to create hierarchies of cortical columns in which most inter-column connections are reciprocal.

4) The internal connectivity within a cortical column is relatively dense and forms multiple feedback loops. Such recurrent circuits and feedback paths are common in artificial recurrent networks implementing sequential/stateful computations, such as associative memory networks Lansner (2009). So, the structure of cortical columns implies that their function is more complex than stateless computations (such as filtering, feature detection or any other memoryless mathematical transformation of their inputs).

5) It has been previously hypothesized, based that cortical columns generate top-down predictions Bastos *et al.* (2012); Rao and Ballard (1999). A column acts as a generative model that can predict its feedforward inputs based on its own priors (stored locally) and also based on predictions that are

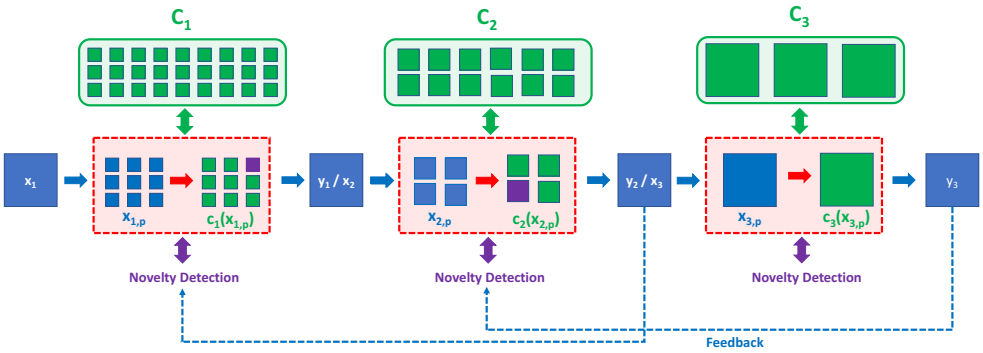


Figure 1: An input image is fed through a 3-layer STAM hierarchy. At each layer, the input is broken up into several overlapping patches (Receptive Fields), which are mapped to centroids using Euclidean distance. If an input RF is flagged to be novel, a new cluster is created and its centroid is initialized based on that patch. Each layer reconstructs its output image based on the selected centroid for each RF, and that image becomes the input to the next layer. Feedback connections are used to control the creation of new centroids based on higher-layer predictions over wider RFs.

fed back from higher-level columns. The STAM model also relies on such top-down predictions, similar to a recent model of cortical columns by Miller Miller (2016).

6) Cortical columns have the capability to incrementally learn from their inputs, storing internal representations that can generalize from few exemplars to useful invariants. For instance, each column of the Inferior Temporal (IT) visual region responds to different orientations or partial views of specific animate or inanimate objects (e.g., faces) Tanaka (1996). Each column is highly selective (e.g., it only responding to faces) but it is also has strong generalization abilities (e.g., responds to the same face independent of rotation, light, occlusions). In other words, it appears that a cortical column stores related "prototypes", and exemplars that are similar to that prototype are recognized by that column Kiani *et al.* (2007); Kriegeskorte *et al.* (2008). From the computational perspective, this is essentially an *online clustering operation*: an input vector is mapped to its nearest cluster centroid (according to some distance metric). Additionally, the centroid of the chosen cluster is adjusted incrementally with every new exemplar so that it comes a bit closer to that input vector - this is how an online clustering module gradually learns the structure of the input data.

7) An online clustering algorithm that is similar to k-means (and asymptotically equivalent to k-means) can be implemented with a rather simple recurrent neural network of excitatory and inhibitory spiking neurons, as shown recently Pehlevan *et al.* (2017). That circuit models the olfactory system in *Drosophila* but similar recurrent E/I circuits are also present in cortical columns.

In summary, a STAM module integrates the following computational functions: intrinsic dimensionality reduction through a multi-layer feedforward hierarchy, online clustering and associative memory formation (i.e., detecting new patterns, learning and updating centroids based on those patterns, and forgetting outlier patterns), and generating top-down predictions. The STAM architecture is described in more detail next.

3 STAM ARCHITECTURE

In its simplest form, a STAM module can be thought of as an online clustering operator that maps its input vector to a nearest centroid of the same dimensionality, also updating the location of that centroid.

A STAM architecture is composed of L layers. The functional unit at each layer is a STAM module. Layer i consists of M_i STAM modules. In the context of object recognition in vision, each STAM processes a Receptive Field (RF) of the input image in that layer. The feedforward input to the m'th STAM module of layer i at time t is denoted by $x_{i,m}(t)$. The set $C_i(t)$ of clusters at layer i is shared among all STAMs of that layer. The j'th centroid of layer i is denoted by the vector $w_{i,j}(t)$. We drop the time variable t when it is not necessary.

Given the set of C_i clusters, each STAM module of layer i selects the nearest centroid to its input based on Euclidean distance:

$$c(x_{i,m}) = \arg \min_{j=1...|C_i|} ||x_{i,m} - w_{i,j}|| \tag{1}$$

The output of layer i, denoted by Y_i, is of the same dimensionality with the input X_i in that layer. Y_i is constructed by the sequence of selected centroids, first replacing the input RF $x_{i,m}$ with the corresponding centroid $c(x_{i,m})$, and averaging the overlapping segments. The input of layer $i+1$ is the output of the previous layer:

$$X_{i+1} = Y_i, \quad i = 1 \dots L-1 \tag{2}$$

We consider square RFs with size $d_i \times d_i$ and stride s_i at layer i, similar to common Convolutional Neural Nets (but this is the only similarity with CNNs, as will be clear next).

3.1 Intrinsic dimensionality reduction

If a d-dimensional binary vector can only take $2^r < 2^d$ values, we say that its intrinsic dimensionality is r while its extrinsic dimensionality is d. A STAM architecture does not need to decrease the dimensionality of its input (even though that can be done if desired). It does reduce however the *intrinsic dimensionality* of its input vector in a gradual manner, as described next.

Suppose that we are given a grey-scale $d \times d$ image $X \in \{0, 1, 2, \dots 255\}^{d \times d}$. The corresponding vectors can take 256^{d^2} possible values, making it hard to reliably cluster them to a much lower-dimensional space that clearly separates the distinct classes these images may belong to. However, we can gradually reduce the intrinsic dimensionality of the input vectors through a multi-layer hierarchy of STAM modules, making such generalization feasible.

At layer i, the size of each RF is $d_i \times d_i$ and the stride is s_i. The number of RFs at layer i (also the number of STAM modules in that layer) is:

$$M_i = (\frac{d - d_i}{s_i} + 1)^2 \tag{3}$$

assuming that $d - d_i$ is a multiple of s_i.

Suppose that we have $|C_i|$ clusters at layer i, and let $r_i = \log_{256} |C_i|$. The input vector $x_{i,m}$ in the m'th RF is clustered into one of $|C_i| = 256^{r_i}$ clusters. The output vector from layer-i is denoted by Y_i and it is formed by overlaying the M_i selected centroids, as described earlier. Since each of the $x_{i,m}$ vectors can take $|C_i|$ values, the vector Y_i can take

$$|C_i|^{M_i} = 256^{r_i \, M_i} = 256^{r_i \, (\frac{d-d_i}{s_i}+1)^2} \tag{4}$$

distinct values. So, the intrinsic dimensionality of the output vector at layer i is $r_i \, (\frac{d-d_i}{s_i} + 1)^2$.

The first layer reduces the intrinsic dimensionality of the input X as long as:

$$r_1 < \frac{d^2}{M_1} = (\frac{d}{\frac{d-d_1}{s_1} + 1})^2 \tag{5}$$

Similarly, layer $i > 1$ reduces the intrinsic dimensionality of its input as long as:

$$r_i < r_{i-1} \, \frac{M_{i-1}}{M_i} = r_{i-1} \, (\frac{\frac{d-d_{i-1}}{s_{i-1}} + 1}{\frac{d-d_i}{s_i} + 1})^2 \tag{6}$$

3.2 Online learning of centroids

A STAM learns in an online manner by updating the centroid that has been selected by its input vector. If the m'th STAM module selected centroid j at layer i for its input vector $x_{i,m}$, we update that centroid as follows:

$$w_{i,j} = \alpha \, x_{i,m} + (1 - \alpha) \, w_{i,j}, \text{when } c(x_{i,m}) = j \tag{7}$$

where the constant α is a learning rate parameter $0 < \alpha < 1$. The higher α is, the learning process becomes faster but also more error-prone. In the rest of this paper, α=0.05.

Centroids are created and initialized dynamically, based on a novelty detection algorithm that is described next.

3.3 NOVELTY DETECTION AND LRU FORGETTING MECHANISM

Any unsupervised continual learning mechanism should be able to quickly detect new patterns that may correspond to new classes. Informally, an input patch $x_{i,m}$ is a "novel" pattern if it is significantly different than its nearest centroid.

To detect novel patterns, we estimate in an online manner the mean distance μ_j between a centroid j and its assigned inputs, updating it with every input $x_{i,m}$ that is assigned to that centroid,

$$\mu_j = \alpha \, ||x_{i,m} - w_{i,j}|| + (1 - \alpha) \, \mu_j \tag{8}$$

Similarly, we estimate in an online manner the standard difference $\hat{\sigma}_j$ (rather than the standard deviation) of those distances from their mean,

$$\hat{\sigma}_j = \alpha \, | \, ||x_{i,m} - w_{i,j}|| - \mu_j \, | + (1 - \alpha) \, \hat{\sigma}_j \tag{9}$$

Both metrics are updated with the same learning rate α we use for the centroid location. The initialization of μ_j requires one additional input, while the initialization of $\hat{\sigma}_j$ requires two additional inputs.

Based on the previous two online estimates, an input $x_{i,m}$ is flagged as "novel" if its distance from the nearest centroid j is significantly larger than the centroid's mean distance μ_j estimate,

$$||x_{i,m} - w_{i,j}|| > \mu_j + 3 \, \hat{\sigma_j} \tag{10}$$

A novel input may correspond to a new pattern that has not been seen before. So, if $x_{i,m}$ is flagged as novel, a new centroid is created at layer i and it is initialized based on that input.

Of course it may be that a novel input is just an outlier that will not be repeated. Thus, it is essential that the architecture also has a mechanism to forget centroids that are not selected frequently enough. Such a *forgetting mechanism* is required if the architecture is to generalize instead of just memorizing its inputs. To provide this mechanism in the STAM architecture, the number of centroids learned at each layer is fixed: layer i cannot remember more than C_i centroids. When that number is exceeded, the centroid that has been *Least Recently Used* (LRU) is forgotten.

Our mechanism for detecting new patterns and forgetting outliers has some similarity with a recent model of novelty detection in the fly olfactory system Dasgupta *et al.* (2018).

3.4 FEEDBACK AND TOP-DOWN PREDICTIONS

The STAM architecture leverages top-down connections (i.e., feedback from the next layer in the hierarchy) to transfer back to the m'th STAM of layer $i < L$ the corresponding m'th RF of the output Y_{i+1} from the next layer. Y_{i+1} has a lower intrinsic dimensionality than Y_i, and it is computed based on a set of centroids C_{i+1} at layer $i + 1$ that cover larger RFs than layer i. For this reason, Y_{i+1} can be viewed as a more general (or prototypical) representation of the input than any single RF of layer i.

Suppose that $y_{i+1,m}$ is the portion of Y_{i+1} that corresponds to the m'th RF at the i layer, and let $c_i(y_{i+1,m})$ be the layer i centroid that is nearest to to $y_{i+1,m}$. This centroid represents the prediction of layer $i + 1$ for the m'th RF at layer i. If the corresponding input $x_{i,m}$ at layer i was flagged as novel but $c_i(y_{i+1,m})$ is one of the existing centroids at layer i, then we do not create a new centroid for that input at layer i. The reason is that, according to the prediction from the next layer, that RF is not truly novel when we consider the entire layer $i + 1$ representation.

3.5 USING LABELED DATA TO ASSIGN CENTROIDS TO CLASSES

The STAM architecture learns in an unsupervised and online manner prototypical patterns (i.e., centroids) seen at different scales of the input data. However, at any point during this continual learning process, we can use these centroids for tasks that are commonly associated with supervised learning, such as classification and object recognition. To do so, we also need a small labeled dataset. Note that the sole purpose of these examples is to assign the already learned centroids to the given set of classes. The labeled examples do not contribute in the learning process.

Here, we describe a simple classifier that first associates each output-layer centroid with a class by calculating the "allegiance" of each labeled input vector x_n to centroid w_j relative to the nearest-neighbor centroid:

$$s_{w_j,x_n} = \frac{e^{-\|w_j - x_n\|}}{\max_{j'} e^{-\|w_{j'} - x_n\|}} \tag{11}$$

The allegiance of centroid w_j to class m is simply the average s_{w_j,x_n} across all labeled inputs x_n that belong to class m:

$$\mathcal{S}_{w_j,m} = \frac{1}{N_m} \sum_{n:y_n=m} s_{w_j,x_n} \tag{12}$$

where N_m is the number of labeled examples of class m, and y_n is the class of input x_n. It is possible that a centroid at the output layer does not have strong allegiance to any class. For this reason, we remove centroids for which the maximum allegiance $\max_m(S_{w_j,m})$ is less than 70% (this threshold was selected by a simple parameter sweep).

The classification of an input x is based on the distance between x and each centroid as well as the allegiance of each centroid to every class. Specifically, x is assigned to the class m that maximizes the following sum across all centroids w_j,

$$k = \arg\max_m \sum_{w_j} S_{w_j,m} e^{-\|w_j - x\|} \tag{13}$$

4 EXPERIMENTS.

We divide the time axis into five learning phases. In each learning phase, the data stream includes two additional classes (digits) from the MNIST dataset, i.e., the first learning phase includes only 0s and 1s, while the fifth learning phase includes all ten digits. In each learning phase, the architecture has access to N_{old} unlabeled examples per class of previously learned classes and N_{new} unlabeled examples per class of newly introduced classes. At the end of each phase, we introduce a limited amount of labeled data per class to evaluate classification accuracy.

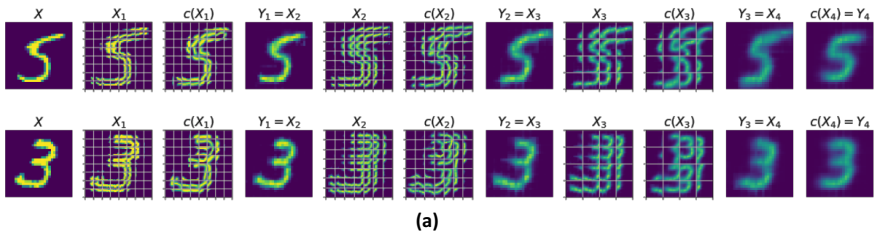
4.1 STAM ARCHITECTURE ANALYSIS

The first experiment visualizes STAM in learning a data stream of handwritten digits Lecun *et al.* (1998) in which the class distribution gradually changes. The data stream is separated into 5 phases, with the first phase consisting of digits $\{0..1\}$, the second phase consisting of digits $\{0..3\}$, and so on, with the fifth phase consisting of digits $\{0..9\}$. In each phase, 1000 digits per class are included in the data stream. At the end of training, we show an example of two images being processed by the final hierarchy (Figure 2.a); as expected, the STAM architecture gradually transforms the input digits into more general (or prototypical) representations of the digit's class.

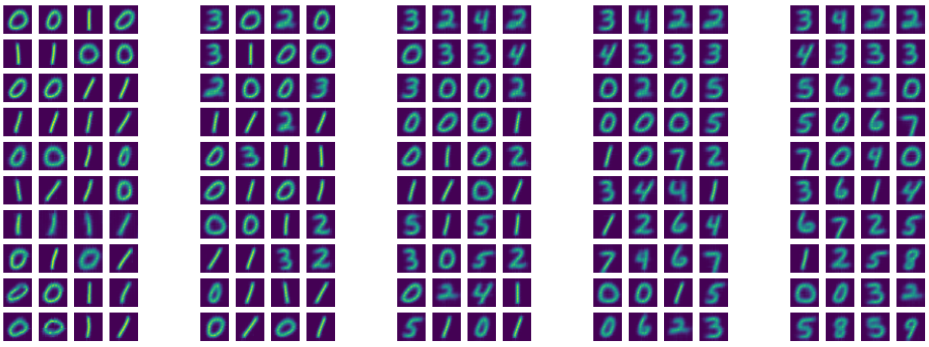
The centroid distributions adapt from phase to phase as the architecture is exposed to classes from a new distribution. In Figure 2.b, we track the 40 most selected centroids in the final layer (layer-4). The STAM architecture can detect the new classes and learn several centroids for both the new and the old classes. Different centroids for the same class correspond to distinct representations of that class.

In earlier layers, small features must adapt to generalize to new classes. In Figure 2.c, a layer-3 centroid is gradually transformed during training. This particular centroid learns a prototypical representation of a common quarter-sphere that is often present in digits 0, 3, 5, 6, and 8.

Our second experiment is aimed to measure catastrophic forgetting in a UCL scenario – and to examine the effect of certain key parameters of the learning process. We evaluate the STAM architecture with different amounts of unlabeled data per class. Each combination consists of N_{old} inputs per class for previously learned classes and N_{new} inputs per class for the newly introduced classes in that learning phase. In Figure 2.d, this is represented with the numbers "N_{old} - N_{new}". We find that as we change the data stream to reduce N_{new}, the architecture is still able to learn new classes even when it only sees 200 unlabeled examples for each new class in a learning phase. Additionally, we find that as we reduce N_{old} (even down to only 200 unlabeled examples per class), the classification



(a)



(b)

(c)

(d)

N_{old} − N_{new}	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5
200 − 1000	99.93 ± 0.17 %	95.34 ± 1.36 %	89.20 ± 1.73 %	88.14 ± 3.29 %	80.26 ± 1.84 %
1000 − 200	100.00 ± 0.00 %	92.26 ± 2.18 %	86.99 ± 1.98 %	88.76 ± 1.57 %	80.07 ± 3.47 %
1000 − 1000	99.93 ± 0.17 %	96.46 ± 0.85 %	90.24 ± 2.12 %	91.32 ± 1.94 %	83.90 ± 1.99 %

(e)

Architecture	1 example per class	3 examples per class	6 examples per class	9 examples per class	12 examples per class
$\|C_i\| = 200$	50.57 ± 6.84 %	66.80 ± 5.00 %	76.82 ± 2.85 %	80.38 ± 1.71 %	83.00 ± 1.77 %
Default	52.71 ± 5.06 %	69.23 ± 4.37 %	78.58 ± 2.95 %	82.19 ± 1.90 %	84.21 ± 1.58 %
$\|C_i\| = 300$	53.12 ± 5.96 %	68.40 ± 5.39 %	79.49 ± 3.67 %	83.10 ± 1.82 %	85.26 ± 1.80 %

Figure 2: (a) Two input images as they go through a 4-layer STAM hierarchy. At layer i, the input X_i is broken up into a matrix of receptive fields, which are mapped to a matrix of centroids ($c(X_i)$) using Euclidean distance. Each layer reconstructs an output image Y_i of the same dimension using the selected centroids, which becomes the input to the next layer. (b) The top-40 layer-4 centroids in five successive phases of a Continual Learning scenario. New classes are gradually introduced into the data stream, starting with $\{0..1\}$ (left) and ending with $\{0..9\}$. (c) The temporal evolution of a specific 13×13 centroid (at layer-3) as it gets updated by inputs that belong to an increasing set of classes. (d) Classification accuracy for data-streams with three different amounts of unlabeled data per class. Each combination consists of N_{old} inputs per class for previously learned classes and N_{new} inputs per class for the newly introduced classes in that learning phase. (e) Classification accuracy for an increasing number of labeled examples, and for three architectures that differ in the number of centroids $|C_i|$ for each layer i.

accuracy does not significantly drop relative to the case that $N_{old} = N_{new} = 2000$; this suggests that the STAM architecture is not subject to major catastrophic forgetting as it learns new classes.

Our third experiment evaluates STAM as a classifier for varying amounts of labeled examples. To simplify, in this experiment all classes are present from the start but the STAM architecture must still learn all classes using online learning and novelty detection. Figure 2.e shows classification accuracy versus number of labeled examples per class for three architectures after seeing 10,000 un-

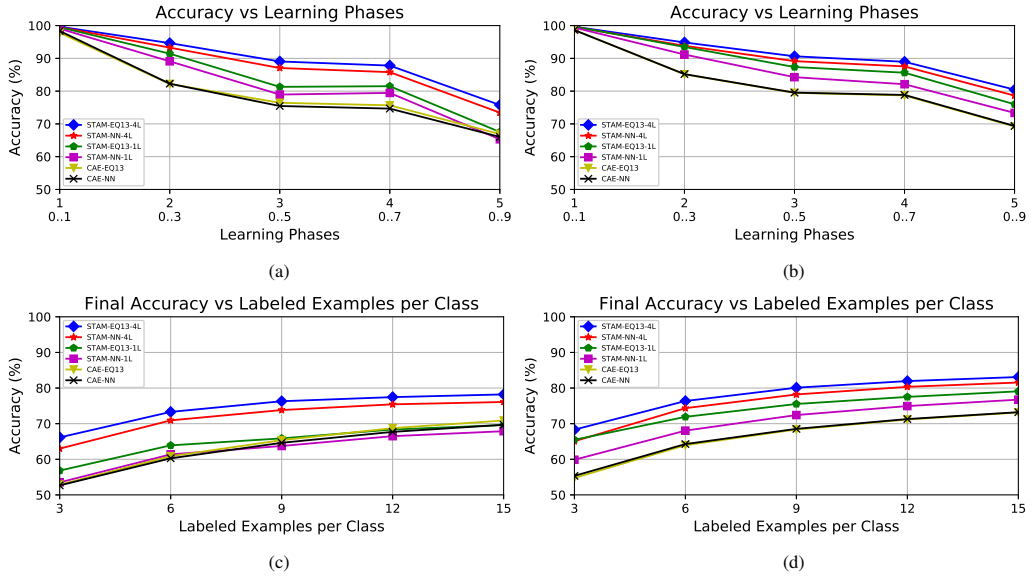


Figure 3: (a) UCL results for $N_{old} = N_{new} = 1,000$ and 10 labeled examples per class (b) UCL results for $N_{old} = N_{new} = 10,000$ and 10 labeled examples per class (c) Final phase accuracy vs. labeled examples per class for $N_{old} = N_{new} = 10,000$ (d) Final phase accuracy vs. labeled examples per class for $N_{old} = N_{new} = 1,000$

labeled inputs. The "default" architecture refers to the architecture of Table 1, whereas "$|C_i|$=200" and "$|C_i|$=300" replace the centroid numbers at each layer with 200 and 300 centroids, respectively. These results demonstrate the capability of the STAM architecture to learn with very few labeled example pairs. Even as few as 6 labeled examples per class are sufficient to result in 80% accuracy. Additionally, the accuracy does not show significant fluctuations as we change the number of centroids at each layer – which is the most important architectural parameter.

4.2 Baseline Comparisons

Together with the STAM architecture, we also train a Convolutional AutoEncoder (CAE) (Li *et al.*, 2017) in an unsupervised manner, and then create a classifier using latent representations of the labeled data for each evaluation period. The CAE architecture was designed specifically for the MNIST dataset, using three convolution and max pooling layers in the encoder and three convolution and upsampling layers in the decoder. We optimize binary cross-entropy loss using the Adam method (Kingma and Ba, 2014). As another baseline, we simply consider a single-layer STAM, which can be interpreted as a non-hierarchical version of the STAM architecture.

For both STAMs and the CAE network, we use two classifiers: nearest-neighbor (NN) and the classifier of (equation 13) – referred to as *EQ13* in the results. We apply EQ13 on the CAE latent representations as centroids with allegiance only to the class corresponding to the input instance's label. The STAM architecture is described in Table 1. We present results (Figure 3) for two experiments on the MNIST dataset (Lecun *et al.*, 1998) based on 10 trials, evaluating accuracy on 10,000 images that were not seen during training.

Table 1: STAM Hierarchy

Layer	RF size	stride	$\lvert C_i \rvert$
1	7	3	200
2	10	3	200
3	13	5	300
4	28	28	400

For our fourth experiment, we compare classification accuracy for various amounts of unlabeled data. We consider $N_{old} = N_{new} = \{1,000,\ 10,000\}$ and provide 10 labeled examples per class for classification. We observe that the performance of the CAE and single-layer baselines strongly fall off when reducing the unlabeled data to $1,000$, whereas the STAM architecture shows less catastrophic forgetting. For our final experiment, we repeat the first experiment varying the amount of labeled data per class and we report only the classification accuracy at the last learning phase. As

expected, STAMs and CAE both see large benefits from increasing the number of labeled examples per class. However, we see that STAM can perform reasonably well with fewer labeled examples compared to the CAE baseline.

5 DISCUSSION

The STAM architecture addresses the desiderata that is often associated with Continual Learning (CL):

1. Online learning: STAMs constantly update their centroids with every example. There is no separate training stage, and there is no specific task for which the network optimizes the features it learns. Any tasks that require classification will of course require one or few labeled examples so that the corresponding clusters that were formed previously are now associated with the name of a class. However, STAMs do not leverage these labeled examples in order to learn features (i.e. STAMs learn in an unsupervised manner).

2. Transfer learning: The hierarchical nature of the proposed architecture means that features learned (in an unsupervised manner) at lower-level STAMs can be reused in different tasks that higher-level STAMs perform. This process is also taking place in the top-down direction: for instance, if the visual data shift at some point from bright to dark images, but the objects are still the same (e.g., animals), the centroids of the higher-level STAMs will remain the same, modulating the lower-level STAMs to darken their centroids instead of learning new prototypes.

3. Resistance to catastrophic forgetting: The introduction of a new class or prototype will lead to the creation of new clusters at some STAMs in the hierarchy (e.g., layer-1 STAMs will learn new elementary visual features if we start feeding them natural images instead of MNIST examples – while a STAM at a higher-level would create a new cluster when it first starts seeing examples of scooters but without affecting the cluster associated with bicycles).

4. Bounded system size: The learning capacity of a STAM architecture depends on two factors: the number of STAMs and the maximum number of centroids that each STAM can store. These two capacity constraints require the system to forget past prototypes that have not been recently updated with new exemplars because the corresponding cluster centroids will gradually shift towards more recently exemplars of different prototypes. This is a graceful forgetting process however (e.g., gradually forgetting the facial characteristics of our children when they were ten years younger).

5. No direct access to previous experience: A STAM only needs to store the centroids of the clusters it has learned so far. Those centroids correspond to prototypes, allowing the STAM to generalize. All previously seen exemplars are discarded.

REFERENCES

Andre M Bastos, W Martin Usrey, Rick A Adams, George R Mangun, Pascal Fries, and Karl J Friston. Canonical microcircuits for predictive coding. *Neuron*, 76(4):695–711, 2012.

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *The European Conference on Computer Vision (ECCV)*, September 2018.

Sanjoy Dasgupta, Timothy C Sheehan, Charles F Stevens, and Saket Navlakha. A neural data structure for novelty detection. *Proceedings of the National Academy of Sciences*, 115(51):13093–13098, 2018.

Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The Helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

R. J. Douglas and K. A. C. Martin. Neuronal circuits of the neocortex. *Annual Review of Neuroscience*, 27:419–452, 2004.

R.J. Douglas, K.A.C. Martin, and D. Whitteridge. A canonical microcircuit for neocortex. *Neural Computation*, 1:480–488, 1989.

Dileep George and Jeff Hawkins. Towards a mathematical theory of cortical micro-circuits. *PLoS computational biology*, 5(10):e1000532, 2009.

Yen-Chang Hsu, Yen-Cheng Liu, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488*, 2018.

Jon H. Kaas. Evolution of columns, modules, and domains in the neocortex of primates. *Proceedings of the National Academy of Sciences*, 109(Supplement 1):10655–10660, 2012.

Matthias Kaschube, Michael Schnabel, Siegrid Löwel, David M. Coppola, Leonard E. White, and Fred Wolf. Universality in the evolution of orientation columns in the visual cortex. *Science*, 330(6007):1113–1116, 2010.

Roozbeh Kiani, Hossein Esteky, Koorosh Mirpour, and Keiji Tanaka. Object category structure in response patterns of neuronal population in monkey inferior temporal cortex. *Journal of neurophysiology*, 97(6):4296–4309, 2007.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

Diederik P. Kingma, Danilo J. Rezende, Shakir Mohamed, and Max Welling. Semi-supervised learning with deep generative models, 2014.

Nikolaus Kriegeskorte, Marieke Mur, Douglas A Ruff, Roozbeh Kiani, Jerzy Bodurka, Hossein Esteky, Keiji Tanaka, and Peter A Bandettini. Matching categorical object representations in inferior temporal cortex of man and monkey. *Neuron*, 60(6):1126–1141, 2008.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

Anders Lansner. Associative memory models: from the cell-assembly theory to biophysically detailed cortex simulations. *Trends in neurosciences*, 32(3):178–186, 2009.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

Fengfu Li, Hong Qiao, Bo Zhang, and Xuanyang Xi. Discriminatively boosted image clustering with fully convolutional auto-encoders, 2017.

William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016.

Kenneth D. Miller. Canonical computations of cerebral cortex. *Current Opinion in Neurobiology*, 37:75–84, 2016.

Takeru Miyato, Shin-ichi Maeda, Shin Ishii, and Masanori Koyama. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

Avital Oliver, Augustus Odena, Colin Raffel, Ekin D. Cubuk, and Ian J. Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. NIPS 2018 Proceedings, 2018.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review, 2018.

Cengiz Pehlevan, Alexander Genkin, and Dmitri B Chklovskii. A clustering neural network model of insect olfaction. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pages 593–600. IEEE, 2017.

Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1):79, 1999.

R. Clay Reid. From functional architecture to functional connectomics. *Neuron*, 75(2):209–217, 2012.

Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks, 2015.

Keiji Tanaka. Inferotemporal cortex and object vision. *Annual review of neuroscience*, 19(1):109–139, 1996.