A bio-inspired incremental learning architecture for applied perceptual problems

Alexander Gepperth[1] and Cem Karaoguz[1]*

Abstract

We present a biologically inspired architecture for incremental learning that remains resource-efficient even in the face of very high data dimensionalities (>1000) that are typically associated with perceptual problems. In particular, we investigate how a new perceptual (object) class can be added to a trained architecture without retraining, while avoiding the well-known catastrophic forgetting effects typically associated with such scenarios. At the heart of the presented architecture lies a generative description of the perceptual space by a self-organized approach which at the same time approximates the neighbourhood relations in this space on a two-dimensional plane. This approximation, which closely imitates the topographic organization of the visual cortex, allows an efficient local update rule for incremental learning even in the face of very high dimensionalities, which we demonstrate by tests on the well-known MNIST benchmark. We complement the model by adding a biologically plausible short-term memory system, allowing it to retain excellent classification accuracy even under incremental learning in progress. The short-term memory is additionally used to reinforce new data statistics by replaying previously stored samples during dedicated "sleep" phases.

1. INTRODUCTION

This contribution addresses the issue of incremental learning for applied robotic scenarios. In particular, we target perceptual learning scenarios where data dimensionalities are typically higher (>1000) than current incremental learning algorithms are able to handle. Incremental learning itself is a notoriously ill-defined term, referring to methods that relax some of the classical assumption of machine learning, namely the availability of training and test data obtained from sampling the same unknown distribution. Instead, incremental learning algorithms may be trained on a set of training data, and then retrained with data sampled from a (more or less) different distribution, potentially multiple times. Incremental learning may also refer to learning algorithms that receive their training samples one by one without knowing their number in advance, instead of processing all examples at the same time, which effectively amounts to changing data distributions as well.

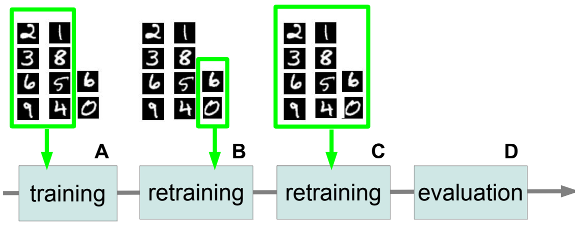

Figure 1. Incremental learning scenario targeted in this study. A) Initial training leaving out a subset of classes (in this paper always just one class). B) optional: incremental retraining with the "missing" classes (for system without short-term memory) C) Re-training with all classes. D) Evaluation of overall classification performance (for all classes) on an independent test set.

In this contribution, we investigate an even stronger form of incremental learning that often occurs in robotic perception problems: the addition of new perceptual classes (e.g., object classes) to a trained classifier. This can happen, for example, by showing a new object to a robot, or a whole set of new objects. This procedure is depicted in Figure 1.

1.1. Problem setting and goals of the study

In perceptual problems, the input dimension I is often quite high, and it is not uncommon for the product IO to exceed 10000 where O denotes output dimension-

*[1] Alexander Gepperth and Cem Karaoguz are with ENSTA ParisTech, UIIS Lab University of Paris-Saclay, 91762 Palaiseau, France and INRIA FLOWERS, 200 avenue de la vielle tour, 33405 Talence, France alexander.gepperth@ensta.fr, cem.karaoguz@ensta.fr

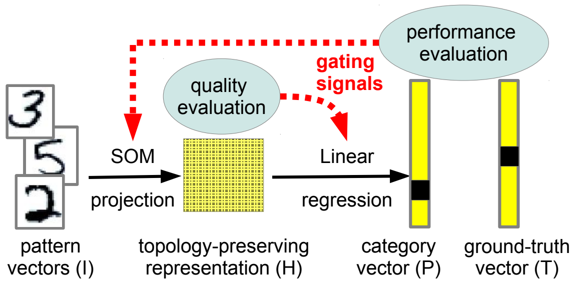

Figure 2. Architecture overview

ality. Furthermore, not only execution but also training time is strongly constrained in a robotics application. Imagine a user teaching a new visual percept to a robot; in this setting, training data must be processed sufficiently quickly to allow interaction, apart from the fact that execution of a trained model must be efficient. Furthermore, the amount of available memory is often severely limited as well if an algorithm is to run on a real robot, potentially on an embedded computer. In addition, in a human-robot interaction scenario, training samples are usually provided sequentially and must be reacted upon immediately, and there is no knowing how many other samples a human might yet provide. Lastly, perceptual problems in robotics are often very challenging ones, so any approach needs to be benchmarked on problems of sufficient complexity.

Therefore, the goal of this study is to present a classification architecture capable of being trained with new classes incrementally as depicted in Figure 1, while being compatible with the following constraints:

- training samples are received one by one
- total number of training samples is unknown
- data dimensionality is high
- execution and training time is limited
- memory is limited
- newly added information should be taken into account immediately
- should work well on challenging and representative real-world tasks

1.2. Related work and taxonomy of incremental learning approaches

Incremental learning comes in various forms in the literature, and the use of the term is not always consistent: first of all, there are approaches while are capable of what we here denote *online learning*, meaning that they take their training samples one by one without knowing their number in advance. Most notably, this is achieved by multilayer perceptrons (MLPs) but there are also extensions of the support vector machine (SVM) model [1, 2] that have this capacity. Online learning is necessary for what we consider to be incremental learning (see list of requirements in the previous section) but not sufficient, as we require as well the capacity to adapt to limited changes (often termed ”concept drift” and/or ”concept shift” [3]) in data statistics while modifying existing models as little as possible. In the neural network literature, it is well-known that such scenarios lead to what is termed ”catastrophic forgetting” [4], the sudden overall deterioration of model performance. To perform true incremental learning in the aforementioned sense, most approaches perform a local partitioning of the input space and train a separate classification/regression model for each partition [5, 6, 7, 8, 9]. The manner of performing this partitioning is very diverse, ranging from kd-trees [9] to genetic algorithms [8] and adaptive Gaussian receptive fields [5]. Equally, the choice of local models varies between linear models [5], Gaussian mixture regression [9] or Gaussian Processes [6]. Since this article is concerned with high-dimensional perceptual problems, it can be stated for all cited approaches that it is really the partitioning of the input space that is costly in terms of memory. Most notably, covariance matrices used in [5] are quadratic in the number of input dimensions which makes their use prohibitive in our problem setting.

1.3. Bio-inspired approach to incremental learning

As biological incremental learning has reached a high degree of perfection, we explicitly investigated the biological literature for hints as how to this might be achieved. Basing ourselves on observations from the basic sensory cortices, we noted that sensory representations seem to be prototype-based, where prototype-sensitive neurons are topologically arranged by similarity [10, 11, 12, 13]. Learning seems to act on these representations in a task-specific way, where more prototypes are allocated to sensory regions where finer discrimination is necessary [14], i.e., where more errors occur during learning. Learning is conceivably enhanced through acetylcholine release in case of task failures [15, 16], leading to higher ”prototype density” in difficult regions of the sensory space. In particular, learning seems to respect and even generate topological layout of prototypes by changing only a small subset of

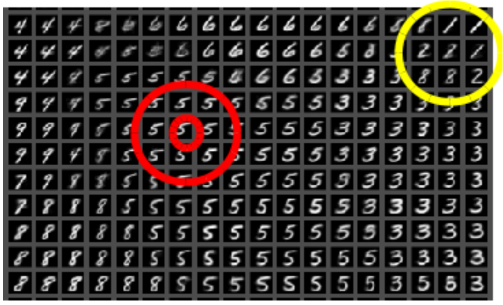

Figure 3. Illustration of how incremental learning is made possible through a topologically ordered prototype representation. Due to topological ordering, neighbouring prototypes are almost always situated in nearby regions of input space. Therefore, local updates of prototypes will almost always be local in input space as well, thus effectively enabling efficient incremental learning. This is shown here for a subset of prototypes trained on the MNIST database, the best-matching unit (BMU) for a ”5” input being indicated by a small red circle. It is obvious that the local 2D update region, indicated by a larger red circle, is indeed local in the input space. The yellow circle indicates a region where this property does not hold (structural defect) but the reader can convince himself that this occurs but rarely.

neural selectivities [17] at each learning event, namely around those neurons that best matched the presented stimulus [13].

We model these findings by using a self-organizing map (SOM) learning to shape the feature preferences of hidden layer neurons in our architecture. SOM is a prototype-based algorithm in the sense that the hidden layer weight vectors ”live” in the space of inputs and aim to approximate the probability density in that space. We model the global, task-related error signal by the current classification error that activates SOM learning in case of mismatch or ambiguity. As SOM learning attributes more prototypes to regions where many learning events occur, this will ensure that prototype density increases in difficult regions of the input space. Furthermore, SOM adaptation is stably self-terminating since no more learning will occur when no more errors are made. Inversely, when error rates increase due to the presentation of new input statistics, the hidden layer representation will become plastic until error rates subside again, when a sufficient re-adaptation has been achieved. Lastly, SOM produces a topologically organized representation of the input space, which was the reason to formulate the model in the first place, and modifies weights only locally in case of learning (as observed in biology). It is above all this localized adaption property that makes us choose the SOM algorithm, as this is an essential prerequisite for incremental learning as will be detailed later in this section. On may speculate that the observed topological organization of selectivities in biological neurons serve just for that: to facilitate incremental learning.

Concretely, we created a three-layer neural network, see Figure 2, that learns a set of plastic, topologically organized prototypes in its hidden layer, implementing the approximation scheme for neighbourhood relations. Consequently, a learning scheme that modifies the best-matching prototype and its (2D) neighbours will be strictly local in input space as well, see Fig. 3. A read-out mechanism between hidden and output layer maps local input space regions (i.e., sets of prototypes) to class memberships using simple linear regression learning. Update of hidden layer prototypes is strictly controlled and occurs only when output layer activity is incorrect or ambiguous, on the grounds that prototypes that allow satisfactory classification need not be adapted. Inversely, the mapping from hidden to output layer is adapted only when there are sufficiently similar prototypes in the hidden layer, since excessive dissimilarity means that unknown data is being fed to the network, and appropriate prototypes must be formed before meaningful readout can occur.

On the computational side, all major approaches to incremental learning, (see Section 1.2), perform a partitioning of the input space and learn independent models in each partition. In this way, learning is always localized in the input space, in the sense that a change of statistics in one part of that space will not affect learning in other, distant parts. It is this property that avoids ”crosstalk” and therefore catastrophic forgetting that is observed in many connectionist models [4]. The presented architecture follows this approach using an approximation scheme that strongly simplifies the definition of local regions which is very costly in spaces of high dimensionality I, for example when using a covariance matrix that will contain I^2 entries. To avoid this, local regions are coarsely approximated by hyperspheres whose centers are defined by support points (or prototypes). The quality of this approximation can be controlled by controlling the overall number of prototypes. As such a *prototype-based representation* approximates the distribution of data points in input space as a whole, it is a generative model [18] as it could be used for sampling purposes. As a local region will usually be defined by more than one prototype, and learning should act on all or at least many prototypes of a region, a way needs to be found to adapt "nearby" prototypes together, amounting to the need of an efficient neighbourhood relation between prototypes. We approximate this by placing prototypes onto a two-dimensional lattice where lattice distance expresses closeness in input space, this property being assured by the learning rule for prototype adaptation. This approximation scheme (which is conveniently and implicitly generated by the SOM learning algorithm), which leads to drastically reduced memory requirements for high-dimensional problems, is explained in Figure 3.

We also incorporate the interplay between short-term and long-term memory in our model. There is a large body of literature investigating the roles of the hippocampal and neocortical areas of the brain in learning. Generally speaking, the hippocampus employs a rapid learning rate with separated representations whereas the neocortex learns slowly, building overlapping representations of the learned task [19]. A well-established model of the interplay between the hippocampus and the neocortex suggests that recent memories are first stored in the hippocampal system and they are played back to the neocortex over time [20]. This accommodates the execution of new tasks that have not been recently performed as well as the transfer of new task representations from the hippocampus (short-term memory) to the neocortical areas (long-term memory) through slow synaptic changes. Inspired from these biological findings, we investigated the effects of such a setup on our model.

Summarizing, we have tried to incorporate as many facts about incremental learning in biology as possible while keeping the model as simple and efficient as possible. Our modeling takes place at the architectural level, leaving aside the finer details of neural modeling (rate/spike code, dynamic neuron models etc.).

2. METHODS

As stated in Section 1.3, we implement our approach to incremental learning as a three-layer neural network architecture depicted in Figure 2. To train the hidden layer H of topologically organized prototypes, we use a learning scheme adapted from the well-known self-organizing map (SOM) algorithm [21] which is described in Section 2.1, whereas the readout from hidden to output layer O is performed by linear regression (LR) explained in Section 2.2. A particular point are the *modulation* influences within the architecture, which control and restrict learning in hidden and output layers, see Section 2.3.

We denote neural activity vector in a 2D representation X by $z^X(\vec{y},t)$, and weight matrices for SOM and LR, represented by their line vectors attached to target position $y=(a,b)$, by $w^{\text{SOM}}_{\vec{y}}$. For reasons of readability, we often skip the dependencies on space and time and include them only where ambiguity would otherwise occur. Thus we write z^X instead of $z^X(\vec{y},t)$ and w^{SOM} instead of $w^{\text{SOM}}_{\vec{y}}(t)$.

2.1. Activity generation and prototype training in the hidden layer

The hidden layer H is not intended to reduce the dimensionality of the input but rather to re-encode it in a way that enables incremental learning, see Figure 3, while preserving information. Therefore, instead of reducing the output of the hidden layer to the best-matching unit (as it is usually done for the SOM model), we calculate the (graded) activations of *all* hidden layer units for performing the following linear regression. Hidden layer activations z^H are meant to measure a sparse similarity between input and the prototype associated to a particular unit and are normalized in the $[0,1]$ interval. They are obtained by first passing all input-prototype distances $\bar{z}^H$ through a Gaussian function with standard deviation κ, and then applying a transfer function $\text{TF}(\cdot)$ that sparsifies these similarities. Here, there is a technical point to be observed: since there is no way of knowing a priori the typical input-prototype distances, κ must be adapted to the data during the learning process, making it a dynamic, time-dependent quantity:$\kappa \equiv \kappa(t)$.

In order not to trigger the adaptation of too many linear regression weights, the transfer function thresholds these similarities, thus ensuring that only those units whose prototypes are very close to the input are active and take part in linear regression training.

Prototype adaptation is performed using the conventional SOM update step except that it takes into account a control signal $\lambda(t)$ coming from the output level of the hierarchy, which will be described in Section 2.3. In precise terms, the equations for activity generation in the hidden layer in response to input activity z^I are as follows:

$$\bar{z}^H(\vec{y}) = ||w^{\text{SOM}}_{\vec{y}} - z^I|| \tag{1}$$

$$\tilde{z}^H = g_\kappa\left(\bar{z}^H\right) \tag{2}$$

$$z^H = \text{TF}_{\theta,p}\left(\tilde{z}^H\right) \tag{3}$$

The adaptation steps for the weights and the distance metric $\kappa(t)$ are as follows:

$$w^{\text{SOM}}_{\vec{y}}(t+1) = w^{\text{SOM}}_{\vec{y}} + \lambda(t)\varepsilon^{\text{SOM}} g_\sigma(||\vec{y}-\vec{y}^*||)(z^I - w^{\text{SOM}}_{\vec{y}}) \tag{4}$$

$$\kappa(t+1) = (1-\tau)\ \kappa(t) + \tau\ \max_{\vec{y}}\bar{z}^H(\vec{y}) \tag{5}$$

range	meaning
$\in [-1, -0.5]$	certain, incorrect
$\in [-0.5, -0.1]$	uncertain, incorrect
$\in [0.1, 0.5]$	uncertain, correct
$\in [0.5, 1]$	certain, correct

Table 1. Significance of various value ranges of the confidence measure $m(z^P)$.

where $g_\sigma(x)$ is a zero-mean Gaussian function with standard deviation s and $\vec{y}^*$ denotes the position of the best-matching unit (i.e., the one with the highest similarity-to-input) in H. In accordance with standard SOM training practices, the SOM learning rate and radius, ε^{SOM} and σ, are maintained at ε_0, σ_0 for $t < T_1$ and are exponentially decreased afterwards in order to attain their long-term values $\varepsilon_\infty, \sigma_\infty$ at $t = T_{\text{conv}}$. TF represents a monotonous non-linear transfer function, $\text{TF} : [0,1] \to [0,1]$ which we model as follows with the goal of maintaining the BMU value unchanged while non-linearly suppressing smaller values:

$$\text{TF}_{\theta,p}(\tilde{z}^H) = \begin{cases} \tilde{\text{TF}}_p(\tilde{z}^H) & \text{if } \tilde{\text{TF}}(\tilde{z}^H) > \theta \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where

$$\tilde{\text{TF}}_p(\tilde{z}^H) = \frac{\left(\tilde{z}^H\right)^p}{\max_{\vec{y}}\left(\tilde{z}^H(\vec{y},t)\right)^{p-1}} \tag{7}$$

2.2. Decision making and learning in the output layer

Generation of output layer activities is performed by a simple linear transformation of thresholded hidden layer activities z^H, using the linear regression weights w^{LR}. Learning is subsequently modifying these weights to optimize the mapping of hidden layer activities z^H to the target representation z^T containing the "true" class of a sample.

$$z^P(\vec{y}) = w^{\text{LR}}_{\vec{y}} \cdot z^H \tag{8}$$

$$w^{\text{LR}}_{\vec{y}}(t+1) = w^{\text{LR}}_{\vec{y}}(t) + 2\varepsilon^{\text{LR}} z^H \left(z^P(\vec{y}) - z^T(\vec{y})\right) \tag{9}$$

In contrast to the hidden layer learning rate, the learning rate of linear regression, ε^{LR} remains constant at all times.

2.3. Feedback control of learning and detection of novelty

Hidden layer and output layer neurons do not adapt their weights all the time, only when it is deemed appropriate. Here, there are two distinct cases to be distinguished: first, when the hidden layer has low overall activity, maybe because the input belongs to a newly added class, then linear regression should not adapt its weights because hidden layer activity is probably not meaningful and it will impair performance for already represented classes. This is achieved automatically by thresholding hidden layer activities in the transfer function $TF_{\theta,p}$ by θ as specified in Equation(3), leading to zero activity if activity is too low. In this case, linear regression weights will not be updated as this requires non-zero activities in the hidden layer H. Secondly, hidden layer weights w^{SOM} will only be updated when the current estimate of class membership, i.e. the output layer activities z^P, is either uncertain or wrong. To measure uncertainty, we first define an uncertainty measure based on the output layer activities, whose basic idea is that a certain estimate of class membership has a clear activity maximum, so a good measure is just to use the bounded difference between first and second maximum:

$$u(z^P) = \max_{\vec{y}} z^P - \max 2_{\vec{y}} z^P \tag{10}$$

This measure, $u(z^P)$, can be combined with the fact whether the activity maximum of z^P is in the right place, i.e., in accordance with ground-truth information z^T:

$$m(z^P, z^T) = \begin{cases} u(z^P), & \text{if arg } \max_{\vec{y}} z^P = \text{arg } \max_{\vec{y}} z^T \\ -u(z^P), & \text{otherwise} \end{cases} \tag{11}$$

Finally, we obtain the modulation measure $\lambda(t)$ that decides whether hidden layer weights should be trained, by thresholding the confidence measure $m(z^P, z^T)$:

$$\lambda(t) = \begin{cases} 0, & m(z^P, z^T) > \theta_m \\ 1, & \text{otherwise} \end{cases} \tag{12}$$

Table 1 gives an overview over the intuitive meaning of various values of the confidence measure m. By thresholding $m(z^P)$ we thus allow hidden layer weights to be trained only when the current class estimate is of less-than-perfect quality, either outright incorrect (for $\theta_m \leq 0$) or correct but of significant uncertainty (for $\theta_m > 0$).

2.4. Incremental learning with short-term memory

We employ a short-term memory mechanism in the form of a simple queue where novel experiences are stored. The novelty of an experience is determined via the comparison of the supervision signal to the prediction z^P of the model which we rename here for convenience to $z^{\text{P-SOM}}$: if the model cannot correctly classify the input (corresponding to $m(z^{\text{P-SOM}}, z^T) < 0$), the feature vector z^I and its label vector z^T are stored in the short-term memory. The short-term memory has limited capacity, hence a new experience overwrites the oldest entry if the short-term memory is full.

In accordance with biological findings, the short-term memory structure is used in both prediction and learning phases. In the prediction phase, response of the short-term memory representation S, $\bar{z}^S$ to an input z^I is computed based on a distance measure, similar to the hidden layer of the model (see Equation 1):

$$\bar{z}^S(i) = ||w_i^{\text{STM}} - z^I|| \tag{13}$$

where w_i^{STM} is the prototype i stored in the short-term memory. The decision whether the model or the short-term memory should be used for final prediction z^{P} is done by comparing the activities $\bar{z}^H$ and $\bar{z}^S$:

$$z^{\text{P}} = \begin{cases} z^{\text{P-STM}}, & \text{if } \min \bar{z}^S < \min \bar{z}^H \\ z^{\text{P-SOM}}, & \text{otherwise} \end{cases} \tag{14}$$

where $z^{\text{P-STM}}$ is the label stored in the short-term memory along with prototype $i^* = \arg\min_i \bar{z}^S$. In this way, the short-term memory shall assist the classification task when the system is exposed to samples from a new class of object (i.e. the incremental learning stage).

Mimicking simulated experiences that occur during sleep [22], contents of the short-term memory are additionally replayed to the model in certain intervals M which are referred to as sleep phases, where the whole content of the STM is replayed to the system, thus training hidden and output layer weights. In this way, novel experiences do not directly cause synaptic changes in the system, they are rather exposed to the model over the short-term memory. This may help the model to quickly build representations for new classes and enable the use of different dynamics for learning new classes.

2.5. Training and testing schedules and dedicated incremental learning measures

The hidden layer training algorithm requires an initialization phase (until T_{conv}, see Section 2.1) and, after that, needs to be trained on $P-1$ classes for a while to provide a starting point for incremental learning. The initial global prototype ordering is terminated at T_1, and hidden layer initialization is performed until T_{conv} without modulation ($\lambda(t) \equiv 1$) as modulation information can only be obtained from a converged hidden layer via the subsequent linear regression stage. Non-incremental training is conducted until T_{inc1} with modulation switched on.

When learning without short-term memory, a first incremental training stage from T_{inc1} to T_{inc2}, and a second stage from then on until the end of the experiment at T_{eval}. For the first incremental training stage, the threshold parameter θ_m, see Section 2.1, is set to a high value: $\theta_m = \theta_m^{\text{inc}}$, whereas it is set to zero for all other training or testing stages: $\theta_m = 0$. This is done to ”protect” the classes that are not in the input data, as a new class would initially lead to quasi-random, weakly localized hidden layer activity which could quickly unlearn previously learned linear regression models. Similarly, the neighbourhood radius $\sigma(t)$, see Section 2.1, is multiplied by a factor of 10 during the first incremental stage, and slowly decreased (back) to its long-term stable value σ_∞ during the second incremental stage. This allows it to perform incremental learning, which requires a large σ for connected regions sensitive to the new class to form in the hidden network layer, while keeping high precision which requires that the final neighbourhood radius σ_∞ be very small.

When learning with short-term memory, a single incremental training stage is conducted from T_{inc1} to T_{eval}, maintaining $\theta_m = 0$ and $\sigma(t) = \sigma_\infty$. Temporal sequence of initialization, training and evaluation steps for the setup without short-term memory are given in Figure 4. For the setup with short-term memory, temporal organization of training and test phases is shown in Figure 5 where the exposition of novel examples is embedded in sleep phases (shown as green periods) and done implicitly. While the scheme illustrated in Figure

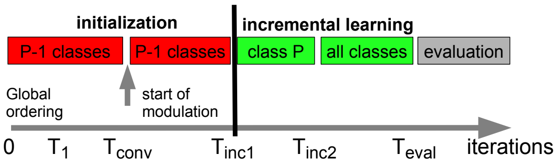

Figure 4. Temporal organization of training and testing phases without short-term memory (x axis not to scale).

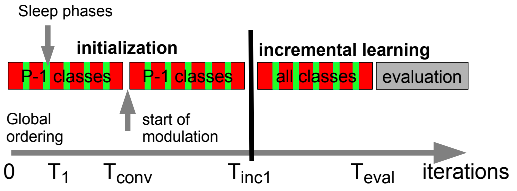

Figure 5. Temporal organization of training and testing phases for the setup with short-term memory (x axis not to scale).

data set	dimensions	# train	# test	preproc.	# classes
MNIST	784	50.000	16.000	none	10

Table 2. Data set used in this study and its principal properties.

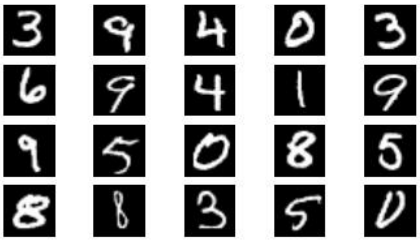

Figure 6. Representative examples from the MNIST data set used in this study.

4 is still viable for the system with short-term memory, blending the incremental learning phase into the whole learning process as in Figure 5 may be more plausible for life-long autonomous learning. All other factors of temporal schedules of training and testing are identical to the setup without short-term memory.

3. EXPERIMENTS

The experiments in this article were conducted on the well-known MNIST handwritten digit recognition benchmark [23], representing a task of moderate difficulty and real-world relevance problem with however a large number (10) of categories. We evaluate performance as indicated by Figure 1: training out model in a leave-one-out fashion on all but one class and the adding the remaining class. We also conduct a final training phase with all classes included before evaluation on the independent test set to draw conclusions about the efficacy of each approach and possible combinations. For a classification task with P classes, this gives essentially P experiments. Please see Table 2 for relevant properties of the used data set, and Fig. 6 for representative data samples.

3.1. Global parameter settings

We use the following fixed parameters for our system, where the total number of neurons in the hidden layer is $n \times n$: $n = 30$, $\varepsilon^{\mathrm{LR}} = 0.001$, $\varepsilon_0 = 0.1$, $\sigma_0 = 0.3n$, $T_1 = 50000$, $T_{\mathrm{conv}} = 150000$, $T_{\mathrm{inc1}} = 800.000$, $T_{\mathrm{inc2}} = 820.000$, $T_{\mathrm{eval}} = 900.000$ $\varepsilon_\infty = 0.001$, $\sigma_\infty = 0.05$, $p = 20$, $\theta_m = 0.0$, $\theta_m^{\mathrm{inc}} = 0.75$, $\theta = 0.75$ and $\tau = 0.001$. SOM, STM and LR weight matrices are initialized to random uniform values between -0.001 and 0.001. Training examples are always randomly and uniformly drawn from the current training set. The capacity of the short-term memory is set to $C = 300$, the system is going into a sleep-phase in every 5000 iterations, replaying all examples stored in the STM.

3.2. Baseline evaluation

To establish a baseline to which we can compare incremental learning performance, we train the proposed architecture on the MNIST data set as described in Section 2, using all classes, while evaluating generalization performance directly after the initialization phase, see Figure 4, yielding a test error rate of 4.8% for MNIST. This is a baseline performance that is quite competitive when compared to other results on this popular benchmark, especially since we are using a rather small hidden layer size of 30x30 elements. With higher hidden layer sizes, the error rate can be improved to $< 2\%$.

3.3. Evaluation of incremental learning performance without short-term memory

Incremental learning experiments on the MNIST data set use the scheme explained in Figure 4. Ten experiments are conducted in total each of which uses one of the 10 classes as the excluded class, labeled Inc-0 through Inc-9. Along with the baseline experiment, Figure 7 shows the evolution of test errors.

As expected, overall error increases as soon as samples from the new class are introduced. The rapid decrease of classification error shows that system learns the newly introduced class quickly. The final error values are usually higher than the values obtained right before the introduction of the new class. However, a slight increase in the overall error should be expected since an added class creates a more difficult classification problem. In addition, we find that the second incremental training step in Figure 1, with all classes included, is unavoidable in almost all cases, a notable exception being the Inc-1 experiment on MNIST shown in Figure 7. This is quite understandable, as a new class in most cases resembles an existing one, thus partly occupies the same volume in input data space and therefore necessarily breaks readout models in this region. This usually concerns just a few classes, as we can see in experiment Inc-0 on MNIST from Figure 7: MNIST classes ”5” and ”6”, which are very similar, are seriously impacted by incremental training of class ”0” but none of the other classes. The additional training step with all classes essentially corrects the balance between classes that are too similar, which is why it improves performance strongly.

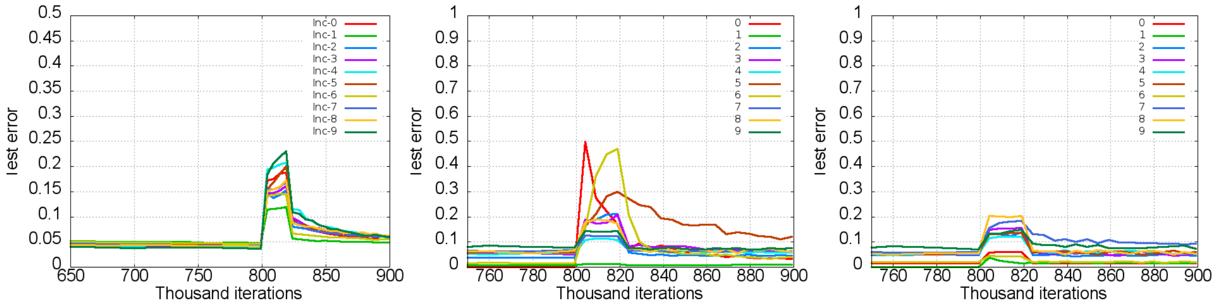

Figure 7. Results of incremental learning experiments Inc-0 through Inc-9 conducted on MNIST. Left: development of overall classification performance for Inc-0 through Inc-9. Middle: development of individual class errors for Inc-0. right: development of individual class errors for Inc-1.

3.4. Qualitative effects and biological analogy of incremental learning

Fig. 8 qualitatively compares the impact incremental learning has on hidden layer prototypes for experiment Inc-0 without short-term memory. We can observe several things: first of all, prototypes of a class are not always homogeneously arranged but can formed several ”islands”, which is normal since prototypes are not organized by class membership but by Euclidean distance. Furthermore, and more surprisingly, prototypes of a class do not always vote most strongly for that class in the linear regression. This is however just an expression of efficiency, as the linear regression uses *all* the information at its disposal: if a prototype that encodes the digit ”1” can be used for classifying the class ”7” as well then it is natural that linear regression will try to do this. And lastly, we can clearly see how class ”0” intrudes into what was previously a region covered by prototypes of class ”6”. When observing the development of classification errors for class ”6” in Fig. 7, we find a corresponding increase for class ”6” during the first phase of incremental learning, reflecting the destruction of many of is prototypes. However, classification performance recovers, probably also because several of the prototypes for ”0” actually help classifying ”6”, as we can plainly see from the color coding.

These results are meant to illustrate the global manner in which incremental learning in the proposed architecture works in practice. They show that new classes are smoothly and strictly locally embedded into existing structures, and always at positions where there is the greatest similarity to existing prototypes, which ensures, as a side effect, that the ”invaded” classes may still profit from the new prototypes as they remain very similar.

3.5. Effects of short-term memory

Experiments with the short-term memory assisted model were done similar to the previous ones. The capacity of the short-term memory is set to $C = 300$, the system is going into a sleep-phase in every 5000 iterations. All other parameters of the model were the same as the previous experiments (see Sec. 3.1). Figure 9 shows the results from experiment Inc-6. The plot on the left shows the frequency of short-term memory usage for prediction throughout the whole experiment. As expected, the system utilizes the short-term memory more often when new experiences are presented: in the beginning and at the start of the incremental learning phase. The impact of the short-term memory in prediction performance becomes apparent if the error plots from experimental setups with and without short-term memory are compared: high peaks of error that occur at the start of the incremental learning phase with the system without short-term memory (Figure 7, middle and right) are suppressed by the utilization of the short-term memory (Figure 9, right).

From the perspective of learning, in some experiments with the system where short-term memory is not utilized, the newly introduced class may cause disruptions in another class that is already learned when the classes share similar features. For example, in experiment Inc-0, the newly added class 0 disrupts already learned class 6 (Figure 7, middle). On the other hand, from the experiments conducted with the model assisted by a short-term memory it can be observed that after the introduction of new samples, test errors per class remain consistent while the model gradually takes over the prediction task from the short-term memory. This suggests that the model successfully learns representations for the new class without breaking the old ones.

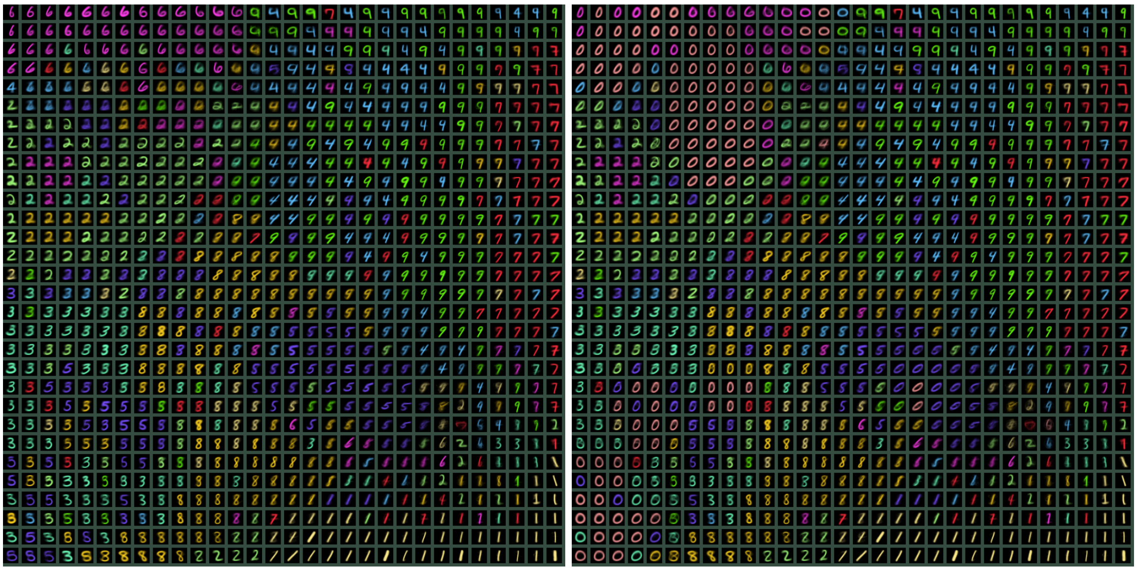

Figure 8. Qualitative effect of incremental learning on hidden layer prototypes for experiment Inc-0 without STM. Shown is a visual representation of prototypes for each of the n^2 hidden layer units. Left: prototypes just before incremental learning of class 0 at $t = T_{\text{inc1}}$. Right: prototypes at $t = T_{\text{inc2}}$. The hue of each prototype corresponds to the class it votes for most strongly, which is read out from the linear regression weights.

4. DISCUSSION

4.1. Significance of contribution

We consider the results of this study very significant in the following sense:

- we present an incremental learning architecture that stays resource-efficient at very high input/output dimensionalities, especially w.r.t. memory consumption, which is not the case for any other incremental learning algorithm we are aware of. For a hidden layer with N^2 elements and input/output dimensionalities of I and P, the memory requirements scale roughly with $N^2(I+P)$ and are thus linear in the sum of input and output dimensions. This means that memory consumption is in no case limited by high problem dimensions but rather by the amount of hidden layer units one is willing to spend for better performance.
- the architecture is conceptually simple, efficient to execute and highly parallelizable as we could show in a previous publication[24]
- we present an intuitive way to benchmark incremental learning and apply it to our architecture using two challenging perceptual problems, with excellent results regarding incremental learning performance, which never degrades overall precision by more than a few percents.
- we present a biologically plausible, efficient and easy-to-implement way of simplifying incremental learning even further using a short-term memory system

4.2. Discussion of methodology

The architecture without short-term memory performs incremental learning successfully but in a rather complicated and ad hoc way. In particular, the system has to know (i.e., be told) when incremental learning is going to happen to that certain parameters (notable θ_m and $\sigma_i nf$) can be set properly for successful incremental learning. While this is a valid approach (as in most applications of incremental learning, e.g., in robotics, it is known when new things must be learned), it is neither very elegant nor very easy-to-use: it would be much more appealing if the system could detect conceptually new samples autonomously, and perform incremental learning on them without the user having to prepare samples beforehand in order to appear exclusively (first incremental training phase). The addition of the short-term memory does just that! It is a very nice side effect that it is in close analogy to biological memory architectures, which may shed some light on the underlying rea-

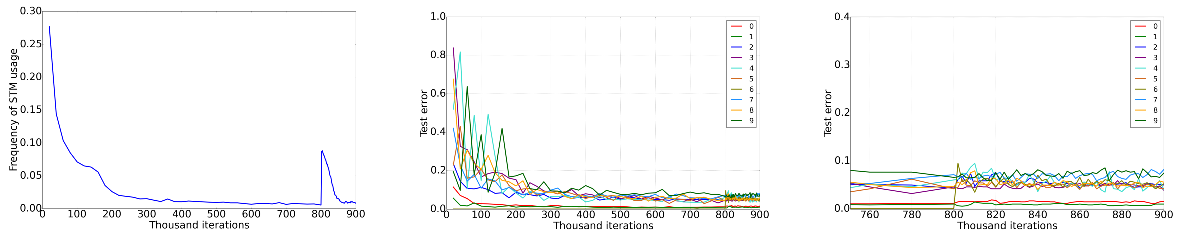

Figure 9. Results of incremental learning experiment Inc-6 with short-term memory conducted on MNIST. Left: frequency of short-term memory usage for prediction. Middle: development of individual class errors for Inc-6. Right: test error of the same experiment zoomed in to the incremental phase for better visualization. The noise in the curves after 800K iterations is due to higher sampling rate of test error.

sons such memory architectures have evolved. From a purely technical point of view, however, the short-term memory solves a concrete need while leaving overall classification performance completely unchanged.

4.3. Comparison to related work

It was attempted to compare our approach to existing prior art in the field of incremental learning, for which we chose the LWPR algorithm[5] following the best practices for choosing parameters as laid down in [25]. However, when working on MNIST data having 756 dimensions and 10 classes, we found that

- performance was unsatisfactory (approximately 15% overall error) when using a single output variable c expressing the class as $c \in \{1,2,3,\dots\}$
- performance improved when using a binary-coded vector of 10 elements, $\vec{c}$, to express class membership. However the number of receptive fields was limited to approximately 15 before freezing the computer (3GHz off-the-shelf PC running Linux, 4GB memory) due to lack of memory. In this case, overall error rates of approximately 7% could be achieved which is well above the error rates achieved by our architecture.

For these reasons, we found that a fair comparison was impossible due to the memory requirements of LWPR when faced with high input and output dimensionalities. As LWPR was not capable to create as many receptive fields as it considered necessary, it could not achieve the best possible performance. We suppose that, using computers with more memory, a fairer comparison could be conducted which will be the subject of subsequent experimental studies. For the time being, we can conclude that LWPR performs much worse than out model in terms of memory consumption when applied to the MNIST dataset, and that its performance, as far as it could be measured, is inferior. We believe that the problem of memory complexity for large input/output dimensionalities will manifest itself for most or all other incremental learning algorithms listed in Sec. 1.2 as all of them perform a partitioning of the input space, often in a fashion that is comparable to the one LWPR uses.

4.4. Conclusion and future work

We have presented a neural architecture for incremental learning that is novel in the sense that it can operate in high-dimensional perceptual spaces. The functionality and feasibility of this architecture has been demonstrated on two real-world perceptual classification benchmarks of sufficient difficulty, showing that very high input/output dimensionalities are no obstacle at all, and that the addition of a new perceptual class to a trained model results in only a very small performance impairment. Furthermore, we included a biologically very plausible version of a short-term memory system, which fulfills very concrete functions: first of all, it allows the system to use a much simpler incremental learning scheme with just a single learning phase. Secondly it permits much quicker ”reactions” to new classes, giving correct classifications almost instantly where otherwise thousands if iterations would be needed for model convergence. And lastly, it enables the system to concentrate learning on samples from new classes that are embedded into samples of known classes: as the new samples arrive relatively rarely, model convergence would be very slow without STM, which was the reason in the first place to split incremental learning into two phases, with new examples submitted exclusively during the first incremental phase.

Future work will include a careful study of the dynamic interplay between long-term and short-term memory, aiming at simplifying the architecture and making as many parameters as possible self-adaptive. This is a prerequisite for doing extended benchmarks on more challenging, real-world classification problems on which the presented architecture should perform well in as generic a fashion as possible.

5. Compliance with ethical standards

This article does not contain any studies with human participants or animals performed by any of the authors. Cem Karaoguz has received a research grant from MBDA Missile Systems. Alexander Gepperth and Cem Karagouz declare that they have no conflict of interest.

References

[1] A. Bordes and L. Bottou. The huller : a simple and efficient online svm. In *Proceedings of the 16th European Conference on Machine Learning (ECML*, 2005.

[2] A. Syed, H. Liu, and K.K. Sung. Incremental learning with support vector machines. 1999.

[3] Pallavi Kulkarni and Roshani Ade. Incremental learning from unbalanced data with concept class, concept drift and missing features: a review. *International Journal of Data Mining and Knowledge Management Process*, 4(6), 2014.

[4] I. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. In *ICLR 2014*, 2014.

[5] S. Vijayakumar and S. Schaal. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high-dimensional spaces. In *International Conference on Machine Learning*, 2000.

[6] D. Nguyen-Tuong and J. Peters. Local gaussian processes regression for real-time model-based robot control. In *IEEE/RSJ International Conference on Intelligent Robot Systems*, 2008.

[7] O. Sigaud, C. Sagan, and V. Padois. On-line regression algorithms for learning mechanical models of robots: A survey. *Robotics and Autonomous Systems*, 2011.

[8] M. Butz, D. Goldberg, and P. Lanzi. Computational complexity of the xcs classifier system. *Foundations of Learning Classifier Systems*, 51, 2005.

[9] T. Cederborg, M. Li, A. Baranes, and P.-Y. Oudeyer. Incremental local online gaussian mixture regression for imitation learning of multiple tasks. 2010.

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

[11] David A Leopold, Igor V Bondar, and Martin A Giese. Norm-based face encoding by single neurons in the monkey inferotemporal cortex. *Nature*, 442(7102):572–575, 2006.

[12] David A Ross, Mickael Deroche, and Thomas J Palmeri. Not just the norm: Exemplar-based models also predict face aftereffects. *Psychonomic bulletin & review*, 21(1):47–70, 2014.

[13] Cynthia A Erickson, Bharathi Jagadeesh, and Robert Desimone. Clustering of perirhinal neurons with similar properties following visual experience in adult monkeys. *Nature neuroscience*, 3(11):1143–1148, 2000.

[14] Daniel B Polley, Elizabeth E Steinberg, and Michael M Merzenich. Perceptual learning directs auditory cortical map reorganization through top-down influences. *The journal of neuroscience*, 26(18):4970–4982, 2006.

[15] Norman M Weinberger. The nucleus basalis and memory codes: Auditory cortical plasticity and the induction of specific, associative behavioral memory. *Neurobiology of Learning and Memory*, 80(3):268 – 284, 2003. Acetylcholine: Cognitive and Brain Functions.

[16] Michael E Hasselmo. The role of acetylcholine in learning and memory. *Current opinion in neurobiology*, 16(6):710–715, 2006.

[17] Edmund T Rolls, GC Baylis, ME Hasselmo, and V Nalwa. The effect of learning on the face selective responses of neurons in the cortex in the superior temporal sulcus of the monkey. *Experimental Brain Research*, 76(1):153–164, 1989.

[18] CM Bishop. *Pattern recognition and machine learning*. Springer-Verlag, New York, 2006.

[19] Randall C Oreilly. The division of labor between the neocortex and hippocampus. *Connectionist Models in Cognitive Psychology*, page 143, 2004.

[20] J. L. McClelland, B. L. McNaughton, and R. 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*, 102:419–457, 1995.

[21] T Kohonen. Self-organized formation of topologically correct feature maps. *Biol. Cybernet.*, 43:59–69, 1982.

[22] Bin Shen and Bruce L McNaughton. Modeling the spontaneous reactivation of experience-specific hippocampal cell assembles during sleep. *Hippocampus*, 6(6):685–692, 1996.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

[24] A Gepperth and M Lefort. Biologically inspired incremental learning for high-dimensional spaces. In *IEEE International Conference on Development and Learning (ICDL)*, 2015.

[25] Sethu Vijayakumar Stefan Klanke and Stefan Schaal. A library for locally weighted projection regression. *Journal of Machine Learning Research (JMLR)*, 9:623–626, 2008.