Biologically inspired incremental learning for high-dimensional spaces
Alexander Gepperth, Thomas Hecht, Mathieu Lefort, Ursula Körner

To cite this version:
Alexander Gepperth, Thomas Hecht, Mathieu Lefort, Ursula Körner. Biologically inspired incremental learning for high-dimensional spaces. Joint IEEE International Conference Developmental Learning and Epigenetic Robotics (ICDL-EPIROB), Sep 2015, Providence, United States. 2015, <10.1109/DEVELRN.2015.7346155>. <hal-01250961>
Biologically inspired incremental learning for high-dimensional spaces

Alexander Gepperth, Thomas Hecht, Mathieu Lefort ENSTA ParisTech/UIIS lab
828 Boulevard des Maréchaux, 91762 Palaiseau, France and
INRIA FLOWERS
200 avenue de la Vieille Tour, 33405 Talence Cedex
Ursula Körner Honda Research Institute Europe GmbH
Carl-Legien-Str.30, 73076 Offenbach am Main, Germany

Abstract—We propose an incremental, highly parallelizable, and constant-time complexity neural learning architecture for multi-class classification (and regression) problems that remains resource-efficient even when the number of input dimensions is very high ($\geq 1000$). This so-called projection-prediction (PROPRE) architecture is strongly inspired by biological information processing in that it uses a prototype-based, topologically organized hidden layer trained with the SOM learning rule that updates hidden layer weights whenever an error occurs. The SOM learning adapts only the weights of localized neural sub-populations that are similar to the input, which explicitly avoids the catastrophic forgetting effect of MLPs in case new input statistics are presented. The readout layer applies linear regression to hidden layer activities subjected to a transfer function, making the whole system capable of representing strongly non-linear decision boundaries. The resource-efficiency of the algorithm stems from the fact of approximating similarity in the input space by proximity in the SOM layer due to the topological SOM projection property. This avoids the storage of inter-cluster distances (quadratic in number of hidden layer) or input space covariance matrices (quadratic in input dimensions) as K-means, RBF or LWPR would have to do. Tests on the popular MNIST handwritten digit benchmark show that the algorithm compares favorably to state-of-the-art results, and parallelizability is demonstrated by analyzing the efficiency of a parallel GPU implementation of the architecture.

I. INTRODUCTION

Incremental learning remains a challenging issue in machine learning. While it is almost self-evident to biologists that learning should be incremental, the technical realization presents baffling difficulties. First of all, incremental learning is inherently sub-optimal when it comes to optimizing an objective (or loss) function. As one can never assume to have seen all training samples at any single point during training, optimization can only take into account the examples seen up to the present moment. Furthermore, the statistics of input-output relations are usually not homogeneous for any finite dataset, so incremental learning must essentially assume non-stationary input statistics at some time scale, which raises the question of how to fuse already learned aspects of a task, without destroying them, with new ones. The latter issue is a real problem for connectionist models of learning [1] and has been termed "catastrophic forgetting", and it is clear that any feasible incremental learning algorithm needs to avoid this issue.

a) Biological foundations and computational modeling: 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 [2], [3], [4], [5]. 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 [6], i.e., where more errors occur during learning. Learning is conceivably enhanced through acetylcholine release in case of task failures [7], [8], 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 neural selectivities [9] at each learning event, namely around those neurons that best matched the presented stimulus [5].

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 in the sense that they are as close as possible to actually occurring inputs according to the SOM energy function (we use a slightly modified SOM model, see[10] that has such a globally decreasing energy function, in contrast to the original model). We model the global, task-related error signal by the current classification error that activates SOM learning in case of mismatch. 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. Thus, the hidden layer represents no longer a pure data distribution but a data distribution modulated by task demands. 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...
followed by linear regression. However, incremental learning with high input and output dimensionalities. On the other dimensionalities input space) has been shown to be very powerful, it runs viewed in [12] partition the input space in a similar way. especially interesting for robotics applications [12], and in population-coded target vector after first applying a non-linear the prototypes that are similar to the best-matching one are organized map (SOM) algorithm [11]. Due to the properties of ing the model as simple and efficient as possible. Our modeling of $K$ of dimensionality $P$, and RFs are created independently for each iteration consists of the following steps, as described in [15], A. The PROPRE architecture

PROPRE is an architecture composed of different algorithmic modules, rather than an algorithm in itself. One PROPRE iteration consists of the following steps, as described in [15], where only the computation of the predictability measure $\lambda$ is changed to represent the current binary classification error: **input:** new data is fed into the input representation $I$ and provided to the SOM, and a new target representation $T$ is provided **projection:** activity is formed in the induced representation $N$ (see Fig. 1) by projection of $I$ onto the SOM prototypes **prediction:** based on activity in $N$, a linear regression step is performed to produce representation $P$ which predicts class membership **evaluation:** a mismatch measure is computed between $P$ and $T$ **update:** linear regression weights are updated. SOM weights are updated only if mismatch was detected. In mathematical terms, the whole model is governed by the following equations, where we denote neural activity at position $\vec{g} = (a, b)$ in a 2D representation $X$ by $z^X(\vec{g}, t)$ and weight matrices for SOM and LR, represented by their $1$All simulation code will be made available for download
line vectors attached to target position $y = (a, b)$ by $w_S^\text{SOM}$:

$$
\tilde{z}^N(y, t) = |w_S^\text{SOM}(t) - z^I(t)| \quad (1)
$$

$$
z^N(y, t) = g_S(\tilde{z}^N) \quad (2)
$$

$$
z^P(y, t) = w^\text{LR}(t) \cdot \text{TF}(z^N(t)) \quad (3)
$$

$$
\lambda(t) = 0 \text{ if } \arg\max_g z^P(y, t) = \arg\max_g z^T(y, t)
$$

$$
1 \text{ else } w^\text{LR}(t + 1) = w^\text{LR}(t) + 2\epsilon^\text{LR} z^I(t) \left( z^P(t) - z^T(t) \right)
$$

$$
w^\text{SOM}(t + 1) = w^\text{SOM}(t) + \lambda(t) \epsilon^\text{SOM} g_S(y, y') (z^I - w^\text{SOM}) \quad (4)
$$

\[ \kappa(t + 1) = 0.999 \kappa(t) + 0.001 \max_g \tilde{z}^N(y, t) \]

where $g_S(x)$ is a zero-mean Gaussian function with standard deviation $s$ and $y'$ denotes the position of the best-matching unit (the one with the highest activity) in $N$. In accordance with standard SOM training practices, the SOM learning rate and radius, $\epsilon^\text{SOM}$ and $\sigma$, start at $\epsilon_0, \sigma_0$ and are exponentially decreased in order to attain their long-term values $\epsilon_\infty, \sigma_\infty$ at $t = T_{\text{conv}}$. In order to convert euclidean distance between a prototype vector and the input into a similarity score in the interval $[0, 1]$, we pass the "naked" distances $\tilde{z}^N$ through a zero-mean Gaussian function with a standard deviation $\kappa(t)$ that adapts to the average maximal distance found in the whole induced representation. This amounts to determining, over the expected maximal distance to which a small score should be assigned, whereas the highest score would always be assigned to a distance of 0. TF represents a monotinous 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 gradually and nonlinearly suppressing all other values:

$$
m_0 = \max_g \tilde{z}^N(y, t)
$$

$$
m_1 = \max_g (\tilde{z}^N(y, t))^{20}
$$

$$
\text{TF} \left( (\tilde{z}^N(y, t) \right) = \frac{m_0 \left( (\tilde{z}^N(y, t) \right)^{20}}{m_1} \quad (5)
$$

A softmax function would have done the trick as well, but we avoid this in order not to calculate too many exponentials.

B. The MNIST handwritten digit database

For all experiments, we use the publicly available MNIST classification benchmark as described in [14]. It contains 10 classes, corresponding to the 10 handwritten digits from "0" to "9", see also Fig. 1, and comes separated into a well-defined train set and a smaller test set. Each sample has a dimensionality of $K = 28 \times 28 = 784$. From the MNIST benchmark, we extract several subsets of classes: $D_0$ containing the digits from 1 to 9, and $D_9$ containing just the digit "0". Analogously, we create $D_1, D_3, D_4$ and $D_5$. Each set $D_6$ is again split into training and test sets $D^{\text{train}}_6, D^{\text{test}}_6$ to measure generalization performance, the split being made according to whether a certain sample belongs to the MNIST train or test set. For training and evaluating performance on all digits, we also create the sets $D^{\text{train}}_{0-9}, D^{\text{test}}_{0-9}$ which correspond to the original MNIST train and test sets.

III. Experiments

We use the following fixed parameters for PROPRE: $\epsilon^\text{LR} = 0.001, \epsilon_0 = 0.5, \sigma_0 = 0.3n, T_1 = 50000, T_{\text{conv}} = 100000, \epsilon_\infty = 0.001$ and $\sigma_\infty = 0.5$. Both SOM and LR weight matrices are initialized to random uniform values between -0.001 and 0.001. No preprocessing is performed on the 28x28-dimensional MNIST input vectors. Training examples are always randomly and uniformly drawn from the current training set.

A. Baseline performance measurement

In order to establish a baseline performance that demonstrates the principal capability of the PROPRE architecture to solve the classification problem posed by MNIST, we first train and evaluate the PROPRE architecture for $10^6$ iterations on $D_{0-9}$. Modulation of SOM learning is turned off by setting $\lambda(t) \equiv 1$ in eqns.(1) as this is not in incremental learning task. The performance thus obtained is to be compared to offline, batch-type algorithms. In the case of PROPRE, this would be other three-layer architectures such as multilayer perceptron or RBF networks. In particular, a goal of this experiment is to find a value of $\epsilon_\infty$ that will give maximal performance in this non-incremental setting. This is an important point as the capacity of the hidden layer to represent inputs as closely as possible is intimately tied to this parameter: the smaller it is, the smaller will be the average prototype-input distance expressed by the similarity score of the BMU, and it can be reasonably speculated that this is in turn related to classification performance of the linear regression readout. We will at the same time vary the hidden layer size $n^2$, again with the goal of maximizing classification performance, where it is again reasonable to suppose that bigger hidden layers will give better classification as the inputs can be approximated with a higher overall resolution.

The results of this set of experiments are summarized in Table I. The show mainly two things:

- Smaller $\epsilon_\infty$ leads to better classification performance
- Bigger hidden layer sizes lead to better classification performance

Especially the latter result, while not really surprising, is interesting, as it suggests that in the case of the PROPRE architecture, we do not suffer from the problem of choosing a correct hidden layer size as in the case of multilayer perceptrons (MLPs). As the hidden layer projections are shaped by an energy-based variant of the generative SOM learning
algorithm, it is intuitively clear that having more prototypes
implies a more precise representation of inputs which in turn
favors classification performance. As a last point, we found,
again not very surprisingly, that the application of a non-
linear transfer function to the SOM similarity scores computed
according to Sec. II-A is essential for acceptable performance.
With purely linear transfer functions, performance drops of
more than 10% occur where precise figures depend on hidden
layer size $n^2$.

B. Incremental learning performance

We conduct several experiments designed to measure the
capability to perform incremental learning. To this effect, we
let the architecture converge on the datasets $D_{\text{train}}^0$, $D_{\text{train}}^1$ or
$D_{\text{train}}^2$ for 300.000 iterations, keeping all timing parameters like
$T_1$ and $T_{\text{conv}}$ unchanged from the baseline experiment, and
performing the same reduction of learning rates and neigh-
bourhood radius. The modulation factor is kept at $\lambda(t) \equiv 1$
for $t < 300.000$ in order to have a defined starting point, and
is determined according to eqns.(1) for $t \geq 300.000$. From
t = 300.000 onwards, we present one of the complementary
datasets $D_{\text{train}}^0$, $D_{\text{train}}^1$ or $D_{\text{train}}^2$ for 20000 iterations, followed
by a phase of 30.000 iterations where SOM plasticity is turned
off ($\lambda(t) \equiv 0$) in order to let linear regression weights "catch
up" with the changes to the hidden layer selectivities. At all
times, we can measure model performance on any of the
corresponding test sets.

e) Results: The numerical results of these experiments,
conducted for $\sigma_{\text{inf}} = 0.05$, are given in Tab. II. They clearly
show that incremental learning is successful, as the newly
added class is well learned while performance on the "old"
classes is retained with little change, which is also reflected
in the fact that the overall error on $D_{\text{train}}^{0-9}$ increases only
insignificantly. As all classes are present in equal frequency in
the MNIST benchmark, an inferior performance on the newly
added class could raise error rates by up to 10 percent which
is not observed.

We observe what happens when the modulation factor $\lambda(t)$
in eqns.(1) is kept fixed at every iteration, regardless of
errors in classification. Although errors on the new class
drop very quickly, errors on the "old" classes rise much
more quickly and unpleasantly than in the case where $\lambda(t)$
is determined from current classification accuracy according
to eqns.(1). This is because eqns.(1) update SOM weights
only when misclassifications occur, or conversely, do not adapt
anything when classifications are accurate. When presenting
a new class, initially all classifications will be incorrect and
strong adaptation occurs. After having learned a sufficiently
good representation in the hidden layer, adaptation of SOM
weights largely stops, which protects the old classes regardless
of how long the new class is actually presented. Briefly put,
the learning architecture we presents adapts its hidden layer
selectivities only as much as necessary and no more.

We furthermore observe that the value of $\sigma_{\text{inf}}$ seems to
control the incremental learning capacity of the architecture:
if it is too large, adaptation is too fast and the old classes will
be completely overwritten before all samples of the new class
have even been fully presented. conversely, if it is too small,
changing input statistics are incorporated too slowly to play a
role within the considered 20.000 retraining iterations. This is
natural since the SOM algorithm guarantees a graceful decay
whose time scale is however controlled by $\sigma_{\text{inf}}$. Fig. 2 shows
the effect of large, small and just correct values for $\sigma_{\text{inf}}$.

C. Parallelization and complexity issues

Apart from the very favorable time and memory complexity of
the PROPRE architecture, all of its component parts can be
very efficiently parallelized. Here we focus predominantly
on the SOM layer as it produces the highest computational
burden. We can deduce from eqn. (1) that both the projection
as well as the weight adaptation part can be performed
in parallel for each output neuron. This is evident for the
projection step that produces $z^N$, as well as for the weight
adaptation step that just makes use of the activities $z^N$ com-
duted during the projection step but not of the weight vectors
(SOM prototypes) of other hidden layer neurons. In fact, in
a parallel implementation we can go even further and stop
weight adaptation for a particular hidden layer neuron when it
is too far away from the best-matching unit (BMU), i.e., when
the neighbourhood function $g_r$ falls below a certain threshold
for which we have taken a very conservative value of $10^{-2}$.
Since the weight change becomes negligible by multiplying
with the current learning rate ($<< 1$) this is a very justi-
ified approximation. For a purely CPU-based implementation
without parallelism, this enormously speeds up computations
when the neighbourhood radius has converged to its long-term
level of $\sigma_{\inf}$, as effectively only a very small fraction of the
total set of prototypes is adapted at each time step, namely
those that are very close to the BMU. For a parallel GPU
implementation, the benefit of this approximation depends

<table>
<thead>
<tr>
<th>train dataset $D_{\text{train}}^i$</th>
<th>$D_{\text{train}}^0$</th>
<th>$D_{\text{train}}^1$</th>
<th>$D_{\text{train}}^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{\text{train}}^0$ at $t = 300.000$</td>
<td>5.95</td>
<td>6.11</td>
<td>5.75</td>
</tr>
<tr>
<td>$D_{\text{train}}^1$ at $t = 350.000$</td>
<td>3.57</td>
<td>0.96</td>
<td>6.5</td>
</tr>
<tr>
<td>$D_{\text{train}}^2$ at $t = 350.000$</td>
<td>6.41</td>
<td>6.75</td>
<td>6.19</td>
</tr>
<tr>
<td>$D_{\text{train}}^{0-9}$ at $t = 350.000$</td>
<td>5.82 (4.8)</td>
<td>5.96 (4.8)</td>
<td>6.21 (4.8)</td>
</tr>
</tbody>
</table>

**TABLE II**
INCREMENTAL LEARNING PERFORMANCE WHEN TRAINING ON 9 MNIST CLASSES AND ADDING THE REMAINING ONE AFTERWARDS, FOR THREE DIFFERENT CHOICES OF THE LATTER, NAMELY "0", "1" AND "2". FOR EACH CHOICE OF RETRAINING DATASET $D_{\text{train}}^i$, ACCURACY IS MEASURED FOR THREE DIFFERENT TEST SETS: $D_{\text{test}}^{\text{conv}}$, $D_{\text{test}}$ AND $D_{\text{test}}^{0-9}$. FOR THE LATTER, THE FIGURE IN PARENTHESES IS THE BASELINE PERFORMANCE, THAT IS, FOR THE CASE OF NON-INCREMENTAL LEARNING.

**TABLE III**
EXECUTION TIMES IN SECONDS PER 10^4 ITERATIONS OF PROPRE FOR GPU AND CPU IMPLEMENTATIONS. IT CAN BE OBSERVED THAT GPU EXECUTION TIMES SCALE (MUCH) LESS THAN LINEARLY IN THE CONSIDERED RANGE OF HIDDEN LAYER SIZES, WHEREAS THE CPU IMPLEMENTATION SCALES ALMOST EXACTLY IN A LINEAR FASHION.
be parallelized, this is not a crucial issue. To show to what extent and with what facility PROPRE can at some point, but as the whole point of this experiments was larger architectures. It is clear that this behavior will saturate grow much more slowly and allow therefore to simulate much the change in hidden layer size, the GPU execution times one hand in absolute terms and on the other hand in terms difference between CPU and GPU implementations, on the summarized in Tab. III. They show a very large performance above was not employed. Obtained execution times are sum-

ized in Tab. III. They show a very large performance difference between CPU and GPU implementations, on the one hand in absolute terms and on the other hand in terms of scaling: whereas CPU execution times scale exactly with the change in hidden layer size, the GPU execution times grow much more slowly and allow therefore to simulate much larger architectures. It is clear that this behavior will saturate at some point, but as the whole point of this experiments was to show to what extent and with what facility PROPRE can be parallelized, this is not a crucial issue. 

IV. DISCUSSION

f) Complexity: Incremental learning using the PROPRE architecture comes at constant time complexity; this is in contrast to conventional incremental learning algorithms such as LWPR which allocate "receptive fields" at runtime as needed, and whose time complexity is linear in the number of these structures. In a sense, PROPRE "pre-allocates" a certain number of "receptive fields" and uses them as well as it can, where having more receptive fields means better classification accuracy. Denoting input dimensionality, hidden layer size and output layer size (number of classes) by K,N and P, the memory complexity of PROPRE is roughly \((KN + NP) = N(K + P) \approx NK\). For a pure RBF classifier with N cluster centers, the memory complexity would be \(N(K + P) \approx NK\) as well. However, if cluster centers should need to be updated in an efficient fashion, it will be necessary to store a matrix of inter-cluster distances so that each new sample can update the clusters to which it is nearest in the input space. This matrix will have \(N^2\) entries, making the total memory complexity \(N(N + K + P \approx N(N + K)\) which can be formidable for a large number of clusters. For the LWPR algorithm, the storage of receptive fields that are defined in the input space requires approximately \(PN(K + 4K^2) \approx 4PNK^2\) which becomes prohibitive for large input dimensionalities \(K\). The factor 4 in the last expression comes from the storage of sufficient data statistics along with each receptive field as detailed by [18].

g) Is this really incremental learning?: in this article, we show how we can, in an additional training step, teach new things to our architecture without forgetting too much of previously learned knowledge. Where forgetting happens, it has a certain graceful decay property that is characteristic for the SOM model. However, in order to teach "new tricks" to the architecture, we perform a dedicated incremental learning procedure that is different from the initial learning procedure: first, we present the new concept in the form of examples, and perform a subsequent linear regression retraining where SOM learning is deactivated. Each time something new (e.g., a class) is added, this step has to be repeated. This does not pose problems in practice, but from a conceptual point of view it would be much more elegant to perform incremental learning identically to the initial learning step. Already, the term "incremental learning" is not well defined in the literature, but if we stick to the terms defined in [12], we can say that our method is incremental but not fully online.

h) Incremental vs. non-incremental performance: We observe in all experiments that incremental learning incurs a slight cost in the form of a departure from the non-
incremental version of the PROPRE architecture. This is not very surprising, first of all as the online linear regression we use to read out hidden layer activities may not yet be fully converged, but mainly because the addition of a new class to an already learned model is inherently sub-optimal (initial learning does not know at all about the new classes).

i) Influence of \(\sigma_{\infty}\): We found a strong impact of \(\sigma_{\infty}\) on map formation in the hidden layer which in turn strongly influences classification accuracy. As suggested by Fig. 2, this parameter also governs the way incremental learning adapts SOM prototypes. We believe that further work will be necessary to elucidate the precise role of this parameter.

V. CONCLUSION

We have presented an algorithm for resource-efficient incremental learning that draws its efficiency from principles of biological information processing and showed that it can easily handle data dimensionalities of 750 entries. We showed that it compares favorably with the state-of-the-art on a standard machine learning benchmark (MNIST) in its non-incremental form, and that good classification accuracy persists when training it incrementally by adding classes to a trained model using the same benchmark. The algorithm is self-limiting and destroys old knowledge only "on demand", due to misclassifications. Due to neural design principles, the whole architecture can be very easily parallelized, obtaining performance gains up to a factor of 8, along with a much nicer scaling behavior when the hidden layer size is increased. Future work will include "deep" PROPRE architectures, namely investigating how the incremental learning capacity can be maintained in such an architecture, as well as efforts to make the PROPRE architecture fully online, which means that training and retraining steps should be conducted in the exact same fashion, thus assuring maximal simplicity and applicability in many different applied scenarios where autonomous, uncontrolled learning is required.

REFERENCES