Computational Advantages of Deep Prototype-Based Learning

Thomas Hecht, Alexander Gepperth

Computational advantages of deep prototype-based learning

Thomas Hecht, Alexander Gepperth*

U2IS, FLOWERS team, INRIA, université Paris-Saclay
828 Blvd des Maréchaux, 91762 Palaiseau cedex, France

Abstract. We present a deep prototype-based learning architecture which achieves a performance that is competitive to a conventional, shallow prototype-based model but at a fraction of the computational cost, especially w.r.t. memory requirements. As prototype-based classification and regression methods are typically plagued by the exploding number of prototypes necessary to solve complex problems, this is an important step towards efficient prototype-based classification and regression. We demonstrate these claims by benchmarking our deep prototype-based model on the well-known MNIST dataset.

Keywords: prototype-based learning, pattern recognition, deep learning, incremental learning

1 Introduction

This study is conducted in the field of *prototype-based machine learning*, and especially regarding the question how to render such machine learning approaches more efficient w.r.t. memory consumption. In prototype-based learning, the probability distribution in data space is not expressed in parametric form but by a learned set of samples, the so-called *prototypes*. Prototype-based machine learning methods were originally motivated by prototype theory from cognitive psychology (see, e.g., [1]) which claims that semantic categories in the human mind are represented by a set of "most typical" examples (or prototypes) for these categories. Well-known prototype-based approaches are the learning vector quantization (LVQ) model [2], the RBF model [3] or the self-organizing map (SOM) model [4]. A very popular prototype-based method in computer vision in is particle filtering [5], where a continuous, evolving probability density function is described and updated as a set of prototypes (here denoted *particles*) whose local density represents local probability density. Prototype-based methods are well suited for incremental learning[6, 7] since prototypes have a very obvious interpretation, and can thus be manipulated easily, e.g., by adding, adapting or removing prototypes (see [8] for a precise definition of incremental learning).

* Thomas Hecht gratefully acknowledges funding support by the "Direction Générale de l'Armement" (DGA) and Ecole Polytechnique.

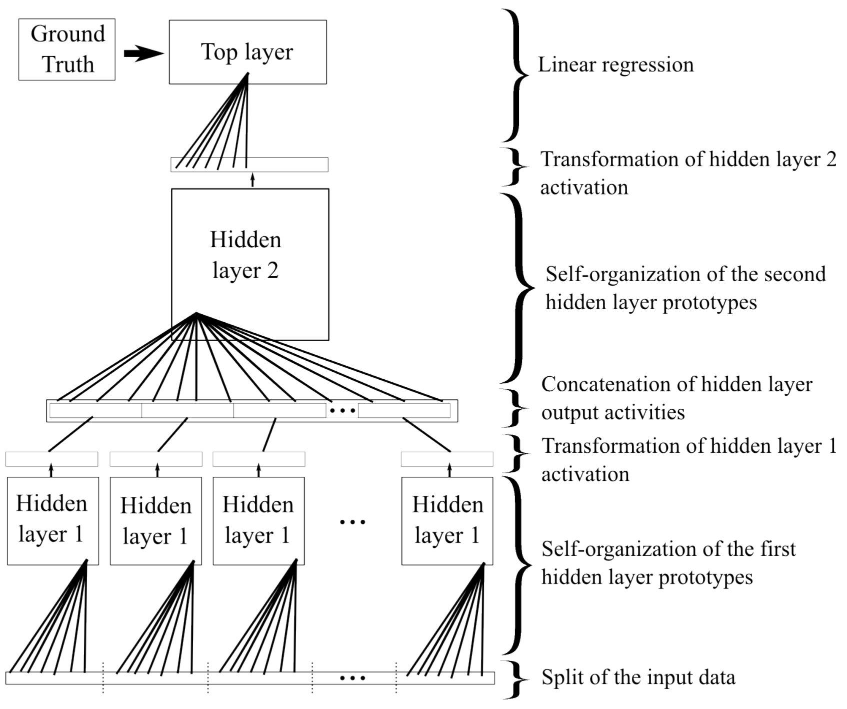
Ground Truth
Top layer
Linear regression
Transformation of hidden layer 2 activation
Hidden layer 2
Self-organization of the second hidden layer prototypes
Concatenation of hidden layer output activities
Transformation of hidden layer 1 activation
Hidden layer 1
Hidden layer 1
Hidden layer 1
Hidden layer 1
Self-organization of the first hidden layer prototypes
Split of the input data

Fig. 1: The hierarchical system used in this study, composed of one input layer, two hidden layers and one top layer.

Prototype-based learning usually has a "flat" architecture (such as the RBF or the LVQ models) with one hidden layer between input and output, where hidden layer weights (the prototypes) describe the input distribution. An obvious problem of such flat architectures is the curse of dimensionality: complex probability distributions in high-dimensional spaces may conceivably require a great number of prototypes to be well approximated, so the memory requirements of flat prototype-based learning can become excessive depending on the problem at hand[9].

This study generalizes "flat" prototype-based learning as presented in [7] to a "deep" architecture (see Fig. 1), with localized receptive fields in the lower layers, just as it is the case in convolutional neural networks (CNNs, see [10]). This makes use of the probabilistic structure of images whose distant parts (receptive fields) are often approximately independent. In this case, it is far more efficient to model their distributions independently as well. If this is done by prototypes, the curse of dimensionality is reversed: as the required number of prototypes can *increase* exponentially with dimensionality, it can also *decrease* exponentially since the dimension of receptive fields is small. In its simplest form, this comes down to a deep four-layer architecture (see Fig. 1), where the first hidden layer now contains prototype activities related to local descriptions of the input, which are subsequently integrated into a global representation in the second hidden layer.

The goal of this study is to show that such a deep prototype-based classifier can achieve a performance that is comparable to its flat counterpart but at a dramatically reduced number of connection weights, which reduces memory consumption and training time. For this purpose, we use the well-known MNIST dataset [11] which is an accepted benchmark in the field of machine learning, and offers the advantage of comparing the absolute performances of both flat and deep architecture to the results of other methods on MNIST.

2 Methods

For representing both first and second hidden layers inputs of the architecture shown in Fig. 1, we use a prototype-based learning algorithm which is loosely based on the self-organizing map model, see [7]. Inputs are represented by graded neural activities arranged in *maps* organized on a two-dimensional grid lattice. Each unit (i, j) of the map X is associated with a weight vector $w_{ij}^X \in \mathbf{W^X}$ which is called *prototype*.

Each map of the M^{h1} first hidden-layer maps has the same size $N_1 = n^{h1} \times n^{h1}$ units and receives a crop from the system input data of size $n^{crop} \times n^{crop}$. M^{h1} is determined by the size of non-overlapping receptive fields in the input layer: the the smaller the receptive fields (each associated with a map in $h1$) is, the greater is M^{h1}. The single map in the second hidden-layer $h2$, of size $n^{h2} \times n^{h2}$, receives a concatenation of activities in the M^{h1} maps of $h1$. The top-layer (output) consists of a linear regression module that computes the prediction $\mathbf{W^{top}}(t)^T \cdot \mathbf{Z^{h2}}(t)$ of the system concerning its current input with $\mathbf{W^{top}}$ the linear regression factors and $\mathbf{Z^{h2}}(t)$ the output activities of the second hidden-layer $h2$.

Because we wish to work in an on-line fashion, weights vectors of all layer are updated at the same time (in contrast to conventional deep architectures which require layer-wise training). Prototypes of the two hidden layers are updated following the Kohonen rule. Each unit's associated weight vector (its *prototype*) is updated using the learning rule and good practices proposed by [4] which decreases learning rate $\epsilon(t)$ and Gaussian neighborhood radius $\sigma(t)$ from initially large values to their asymptotic values ϵ_∞, σ_∞. For each iteration step $t <= T$, prototypes of maps ($\mathbf{W^X}$) are updated depending on the current input $\mathbf{x}$ with the following rule :

$$\mathbf{W^X}(t+1) \leftarrow \mathbf{W^X}(t) + \epsilon(t) \cdot \mathbf{\Phi}(t) \cdot (\mathbf{x}(t) - w^{X^\star}(t)) \tag{1}$$

where ϵ is the learning rate and $\mathbf{\Phi}(t) = \phi(w^{X^\star}(t), \sigma(t))$ is a discretized Gaussian kernel with σ variance, centered on the current best-matching $w^{X^\star}$ unit and representing the neighborhood influence of the weights adaptation.

For any map X, the *map activity* $z_{ij}^X \in \mathbf{Z^X}$ at position (i, j) is then derived from the Euclidean distance between the unit prototype w_{ij}^X and the current input $\mathbf{x}$:

$$z_{ij}^X(t) = \mathrm{f}\left(g_\kappa\left(||w_{ij}^X(t) - \mathbf{x}(t)||\right)\right) \tag{2}$$

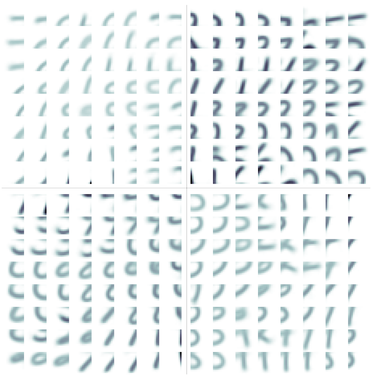

Fig. 2: An overview of prototypes in 4 maps of the first hidden layer $h1$ corresponding to 2×2 receptive fields of 14×14 pixels over MNIST inputs.

where, as described in [7], $g_\kappa(\cdot)$ is a Gaussian function with an adaptive parameter κ that converts distances into the $[0, 1]$ interval, and $\mathrm{f}(\cdot)$ is a monotonous non-linear *transfer function*, defined as :

$$\begin{aligned} m_0 &= \max_{\boldsymbol{y}} \bar{z}^P(\boldsymbol{y}, t) \\ m_1 &= \max_{\boldsymbol{y}} \left(z^P(\boldsymbol{y}, t)\right)^{20} \\ \mathrm{f}\left((z^P(\boldsymbol{y})\right) &= m_0 \frac{\left(z^P(\boldsymbol{y})\right)^{20}}{m_1} \end{aligned} \tag{3}$$

The top-layer weights vector is updated following an on-line stochastic gradient descent mechanism with a fixed learning rate η over time by comparison of its own prediction and the current ground truth label of the current input data. Because we apply our system to classification tasks, labels $\mathbf{y}$ and predictions $\mathbf{p}$ are not scalars but vectors of c values where the position of the maximum indicates respectively the target class label and the system decision (population coding).

$$\mathbf{W}^{\mathbf{top}}(t+1) \leftarrow \mathbf{W}^{\mathbf{top}}(t) - \eta \cdot (\mathbf{y}(t) - \mathbf{p}(t)) \cdot \mathbf{Z}^{\mathbf{h2}}(t) \tag{4}$$

We use two indicators to assess performance: mean classification errors and total number of connection weights.

Mean classification error This indicator is a commonly used measure for comparing performances of a classification system. During the testing phase, when all the learning processes (i.e. weights updates) are stopped, classification errors are logged on the MNIST test set following the simple rule : $\gamma = 0.$ if argmax($\mathbf{y}(t)$) equals argmax($\mathbf{p}(t)$) and $\gamma = 1.$ otherwise. Then, we just compute the mean for all these errors $\mu = \frac{1}{T^{test}} \cdot \sum_{t=1}^{T^{test}} \gamma$.

Total number of weighted connexions This indicator is a simple means to compare our hierarchical architecture and a "flat" architecture, only composed of a huge map and a linear regression module, in terms of number of connexions. Let M^{h1} be the number of maps in the first hidden layer each composed of $N_1 = n^{h1} * n^{h1}$ units and $M^{h2} = 1$ the number of maps in the second hidden layer, composed of $N_2 = n^{h2} * n^{h2}$ units. Because connexions between the second hidden layer and the top layer are negligible in the two architectures, we do not take them into account. The total number of weighted connexions is then computed as : $K = N_1 * (d + N_2 * M^{h1})$ for a hierarchical architecture and $\tilde{K} = d * N_2$ for a "flat" one. By example, an architecture with 4 hidden maps of 4×4 units receiving crops extracted from 784 pixels images (such as MNIST images), followed by a hidden map of 10×10 units followed by the linear regression module gives $K = (4 \times 4) * (784 + (10 \times 10) * 4) = 18'944$. For a comparable "flat" system without the first hidden layer, we would need $\tilde{K} = 784 * (10 \times 10) = 78'400$ connexions.

3 Experiments

3.1 Protocol

Each experiment run consists of $T^{train} = 1'000'000$ training iterations followed by $T^{test} = 50'000$ testing iterations. For all self-organizing maps, we use exponentially decreased values of learning rate and neighbourhood radius and a fixed linear regression learning rate : $\epsilon_0 = 0.25$, $\epsilon_\infty = 0.001$, $\sigma_0 = 0.5 * n^X$, $\sigma_\infty = 0.085$ and $\eta = 0.009$. Both self-organizing maps and linear regression weight vectors are initialized to random uniform values between -0.001 and 0.001. Samples are always randomly and uniformly picked and are provided to the system as input data $\mathbf{x}$ and ground truth y. Results presented below are averaged measures over 10 runs. Datasets targets are split into c different classes and each input data is d dimensional.

We use the publicly available MNIST classification benchmark as described in [11]. It contains $c = 10$ classes, corresponding to the 10 handwritten digits from "0" to "9" and comes separated into a well-defined train set and a smaller test set. Each sample has a dimensionality of $d = 28 \times 28 = 784$.

3.2 Results

As shown in Table 1, our hierarchical system with 4 hidden layer maps can achieve comparable performances with less connexions involved. When comparing mean classification errors in the hierarchical case (μ) and in the "flat" one ($\tilde{\mu}$), it seems that, with respect to the same parameters set, there is no need to add hidden layers. But when looking at the total number of weighted connexions, K is always smaller than $\tilde{K}$: by example, with $M^{h1} = 4$ maps of $N_1 = 6 \times 6$ units in the first hidden layer and one map of $N_2 = 20 \times 20$ unit in the second hidden layer, after T^{train} on-line iterations the hierarchical system can achieve, in average, equivalent classification performances than a system only composed of a single 20×20 self-organizing map but with three times less connexions.

If we try with another number of hidden layer maps - by instance with $M^{h1} = 16$ and $n^{crop} = 7$ - it seems that performances drop down and that the ratio $K/\tilde{K}$ is no longer a real advantage. Because we are dealing with raw pixels and no extracted features on this dataset, the system is extremly sensible to the size of the receptive fields. It seems to us that there is an interessant research question about the well suited n^{crop} : what is the link with the dataset distribution, are overlapping receptive fields a good idea or can the system adapt itself this parameter during the incremental learning paradigm ?

Table 1: MNIST mean classification errors

$n^{crop} = 14$, $M^{h1} = 2 \times 2$					
n^{h1}	n^{h2}	μ	$\tilde{\mu}$	K	$\tilde{K}$
4	10	10.8 (±0.7)	9.1 (±0.5)	18'944	78'400
6	10	9.4 (±0.8)	9.1 (±0.5)	42'624	78'400
8	10	11.2 (±0.6)	9.1 (±0.5)	75'776	78'400
4	20	6.7 (±0.4)	6.3 (±0.6)	38'144	313'600
6	20	**5.9 (±0.3)**	6.3 (±0.6)	85'824	313'600
8	20	6.4 (±0.6)	6.3 (±0.6)	152'576	313'600

$n^{crop} = 7$, $M^{h1} = 4 \times 4$					
n^{h1}	n^{h2}	μ	$\tilde{\mu}$	K	$\tilde{K}$
4	10	12.9 (±0.9)	9.1 (±0.5)	38'144	78'400
6	10	11.1 (±0.8)	9.1 (±0.5)	85'824	78'400
8	10	11.0 (±0.9)	9.1 (±0.5)	152'576	78'400
4	20	10.5 (±0.6)	6.3 (±0.6)	114'944	313'600
6	20	10.1 (±0.5)	6.3 (±0.6)	258'624	313'600
8	20	10.8 (±0.5)	6.3 (±0.6)	459'776	313'600

4 Discussion, conclusion, perspectives

This article has shown that a deep prototype-based architecture is capable of achieving performances comparable to those of a flat architecture of the same type, while drastically reducing the number of connection weights, and therefore memory usage and processing time. We believe that this effect may be observed for any prototype-based method (notably LVQ) if approximatye independence relations hold between separate parts of the input. As stated in Sec. 1, this is almost always the case if inputs are visual images, although of course the right parameters have to be found in the form of receptive field sizes and overlaps. However to be fair, this parameter search would also have to be performed for convolutional neural network (CNN) and is a property of all deep architectures based on local receptive fields.

Given that prototype-based methods in machine learning have a number of highly desirable properties, such as online and incremental learning capacity [7, 6], a simple probabilistic interpretation [12] and a natural way of processing multi-class problems, the reduction of resource requirements even when treating complex visual problems seems an important step towards wide-spread use of prototype-based machine learning methods.

References

1. E. Rosch. Cognitive reference points. *Cognitive Psychology*, 7, 1975.
2. M Biehl, A Ghosh, and B Hammer. Dynamics and generalization ability of lvq algorithms.
3. J. Moody and C. J. Darken. Fast learning in networks of locally tuned processing units. *Neural Computation*, 1, 1989.
4. T Kohonen. Self-organized formation of topologically correct feature maps. *Biol. Cybernet.*, 43:59–69, 1982.
5. M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *Signal Processing, IEEE Transactions on*, 50(2):174–188, 2002.
6. Viktor Losing, Barbara Hammer, and Heiko Wersing. Interactive Online Learning for Obstacle Classification on a Mobile Robot. IEEE, 2015.
7. A Gepperth and C Karaoguz. A bio-inspired incremental learning architecture for applied perceptual problems. *Cognitive Computation*, 2015. accepted.
8. A Gepperth and B Hammer. Incremental learning algorithms and applications. In *European Sympoisum on Artificial Neural Networks (ESANN)*, 2016.
9. Brian Carse and Terence C. Fogarty. *Parallel Problem Solving from Nature — PPSN IV: International Conference on Evolutionary Computation — The 4th International Conference on Parallel Problem Solving from Nature Berlin, Germany, September 22–26, 1996 Proceedings*, chapter Tackling the "curse of dimensionality" of radial basis functional neural networks using a genetic algorithm, pages 707–719. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.
10. Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3626–3633, 2013.
11. Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
12. Petra Schneider, Michael Biehl, and Barbara Hammer. Hyperparameter learning in probabilistic prototype-based models. *Neurocomputing*, 73(79):1117 – 1124, 2010. Advances in Computational Intelligence and Learning17th European Symposium on Artificial Neural Networks 200917th European Symposium on Artificial Neural Networks 2009.