Citation for published version:
Exarchakis, G, Bornschein, J, Sheikh, A-S, Dai, Z, Henniges, M, Drefs, J & Lücke, J 2019 'ProSper -- A Python Library for Probabilistic Sparse Coding with Non-Standard Priors and Superpositions' arXiv.

Publication date:
2019

Link to publication

University of Bath

Alternative formats
If you require this document in an alternative format, please contact:
openaccess@bath.ac.uk

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 18. Jan. 2024

ProSper - A Python Library for Probabilistic Sparse Coding with Non-Standard Priors and Superpositions

Georgios Exarchakis* georgios.exarchakis@inserm.fr
Sorbonne Université, INSERM, CNRS, Institut de la Vision, Paris, France

Jörg Bornschein*,† j.bornschein@gmail.com
FIAS, Goethe-Universität Frankfurt am Main, Germany

Abdul-Saboor Sheikh sheikh.abdulsaboor@gmail.com
Zalando Research, Berlin, Germany

Zhenwen Dai zdai@sheffield.ac.uk
Dept of Computer Science, University of Sheffield, UK

Marc Henniges marc.henniges@d-fine.com
d-fine GmbH, Frankfurt, Germany

Jakob Drefs jakob.drefs@uol.de
Machine Learning Division, C. v. Ossietzky Universität Oldenburg, Germany

Jörg Lücke joerg.luecke@uol.de
Machine Learning Division, C. v. Ossietzky Universität Oldenburg, Germany

Editor: Unknown

*contributed equally, †now at DeepMind, London, UK

arXiv:1908.06843v1 [eess.SP] 1 Aug 2019

Abstract

`ProSper` is a python library containing probabilistic algorithms to learn dictionaries. Given a set of data points, the implemented algorithms seek to learn the elementary components that have generated the data. The library widens the scope of dictionary learning approaches beyond implementations of standard approaches such as ICA, NMF or standard L_1 sparse coding. The implemented algorithms are especially well-suited in cases when data consist of components that combine non-linearly and/or for data requiring flexible prior distributions. Furthermore, the implemented algorithms go beyond standard approaches by inferring prior and noise parameters of the data, and they provide rich a-posteriori approximations for inference. The library is designed to be extendable and it currently includes: Binary Sparse Coding (BSC), Ternary Sparse Coding (TSC), Discrete Sparse Coding (DSC), Maximal Causes Analysis (MCA), Maximum Magnitude Causes Analysis (MMCA), and Gaussian Sparse Coding (GSC, a recent spike-and-slab sparse coding approach). The algorithms are scalable due to a combination of variational approximations and parallelization. Implementations of all algorithms allow for parallel execution on multiple CPUs and multiple machines for medium to large-scale applications. Typical large-scale runs of the algorithms can use hundreds of CPUs to learn hundreds of dictionary elements from data with tens of millions of floating-point numbers such that models with several hundred thousand parameters can be optimized. The library is designed to have minimal dependencies and to be easy to use. It targets users of dictionary learning algorithms and Machine Learning researchers.

Keywords: Python, parallel computing, software library, expectation-maximization, sparse coding, feature learning, latent variable models, variational approximations

1. Introduction

Dictionary learning is a broad subfield of Machine Learning with numerous applications in different data domains. It addresses the unsupervised extraction of latent components or factors of observed data samples. Classical examples of dictionary learning methods include deterministic approaches such as K-SVD, ICA, projection pursuit, NMF among many others. In contrast to deterministic approaches, probabilistic methodologies for dictionary learning are based on a generative data model to yield a probabilistic objective (typically the data likelihood) for optimization. While some probabilistic approaches such as sparse coding with a Gaussian noise model and Laplace prior (Olshausen and Field, 1996) closely link to popular deterministic L_1-regularized sparse coding, many other choices of priors do not have a corresponding counterpart. Similarly, it is straight-forward to define non-standard probabilistic data models, e.g., by choosing the component superposition assumption to be different from linear, which can be a more reasonable choice for many types of data (Bornschein et al., 2013; Dai et al., 2013; Sheikh et al., 2019). In several contributions, it was shown over the past years that such non-standard sparse coding models can efficiently be trained at large scales and with large data sets. Furthermore, parameters other than basis functions (i.e., dictionary elements) can be learned using recent approximate learning methods, notably the sparsity level and the data noise. The `ProSper` library contains such novel and non-standard sparse coding algorithms in a unified python software framework. Most notably, the used priors are binary, ternary, categorical or follow a spike-and-slab distribution, and the superposition models of components are linear or non-linear. Based on truncated posterior approximations (Lücke and Eggert, 2010) and MPI parallelization for many CPU nodes and cores, all algorithms can be efficiently applied to large data sets and large dictionary sizes (Henniges et al., 2010; Guiraud et al., 2018; Exarchakis et al., 2012; Exarchakis and Lücke, 2017; Lücke and Sheikh, 2012; Sheikh et al., 2014; Puertas et al., 2010; Sheikh et al., 2019; Bornschein et al., 2013). The software uses the `NumPy` and `SciPy` packages to define data and parameter containers and apply elementary numerical operations. `ProSper` algorithms are also enabled for parallel computing using the `MPI for Python` package and a data logging utility built on top of the `PyTables` package.

2. Learning Algorithm and Data Models

All the models included in `ProSper` are based on the following data generative process:

$$p(\vec{s}\,|\,\Theta) \;=\; \text{latent variable prior distribution, e.g. Bernoulli for BSC} \tag{1}$$
$$p(\vec{y}\,|\,\vec{s},\Theta) \;=\; p(\vec{y};\, \vec{f}(\Theta,\vec{s})) \quad \text{(noise model)}, \tag{2}$$

where Θ is the set of model parameters (typically containing the dictionary W but also prior and noise parameters). The models can be fully specified by defining Eq. 1 and 2, i.e., they are categorized based on the noise distribution, the prior distribution, and the function $\vec{f}$ that determines the influence of the latent variables on the observed variables ($\vec{f}$ can be thought of as a link function). The most common instance of the function $\vec{f}$

Table 1: List of algorithms with their superposition models for component combination and their assumed distributions for observed and hidden variables.

Model	**Properties**				
Acronym	superposition	obs. variables	noise type	hidden variables	prior model
BSC	linear	real	Gaussian	binary {0,1}	Bernoulli
TSC	linear	real	Gaussian	ternary {-1,0,1}	categorical/zero-mean
DSC	linear	real	Gaussian	discrete	categorical
GSC	linear	real	Gaussian	real	spike-and-slab
MCA	max	real	Gaussian	binary {0,1}	Bernoulli
MMCA	$\vert$ max $\vert$	real	Gaussian	binary {0,1}	Bernoulli
GMM	none	real	Gaussian	one integer	categorical
PMM	none	integer (≥ 0)	Poisson	one integer	categorical

Table 2: List of Prosper models together with their associated references (main ref. bold).

Model	Full Name	References (bold for main reference)
BSC	Binary Sparse Coding	Lücke & Eggert *JMLR* 2010 **Henniges et al., *Proc. LVA/ICA* 2010** Guiraud et al., *GECCO*, 2018
TSC	Ternary Sparse Coding	**Exarchakis et al., *Proc. LVA/ICA* 2012**
DSC	Discrete Sparse Coding	**Exarchakis et al., *Neural Comp.* 2017**
GSC	Gaussian Sparse Coding (spike + Gaussian slab)	Lücke & Sheikh, *Proc. LVA/ICA* 2012 **Sheikh et al., *JMLR* 2014**
MCA	Maximal Causes Analysis	Lücke & Sahani, *JMLR* 2008 Lücke & Eggert *JMLR* 2010 **Puertas et al., *NIPS* 2010** Sheikh et al., *PLOS Comp. Bio.* 2019.
MMCA	Max Magnitude MCA	**Bornschein et al., *PLOS Comp. Bio.*, 2013**
GMM	Gaussian Mixture Model	standard EM for GMM algorithm
PMM	Poisson Mixture Model	standard EM for a Poisson mixture

for dictionary learning is a linear function, e.g. $\vec{f}(\Theta, \vec{s}) = W\vec{s}$, but the library allows for alternative choices. Tab. 1 lists the data models of the currently implemented algorithms.

All algorithms use expectation maximization for parameter optimization and truncated posteriors as efficient approximation (Lücke and Eggert, 2010). The approximation method has been successfully applied in numerous contexts to the models listed in Tab. 1. A list of scientific publications describing the models along with their specific implementation details for inference and learning can be found in Tab. 2.

3. User Interface and Documentation

The interface is designed to be as reusable and flexible as possible. We use three objects that compose a learning algorithm: `Annealing`, `Model`, and `EM`. The learned parameters are contained in a python dictionary that is shared among these objects. The `Annealing` object is responsible for the schedule of the learning algorithm and defines methods `next`, and `reset` to specify the step and technical interventions of the training process. The `Model` object, probably the most crucial element of our library, defines methods `step` and `standard_init` which respectively define one optimization step of the algorithm, and the initialization of the parameters. The `run` method of the `EM` object combines `Annealing` and `Model` by running a loop over the optimization step of the model modified as specified in the `Annealing` object.

Annealing schedules are specified by objects that inherit from the abstract `Annealing` class. As an example, we provide the class `LinearAnnealing` that controls the number of iterations of the training algorithm, parameter noise, and deterministic annealing of the approximate posterior. Similarly, `Model` instances inherit from the abstract `Model` class and implement the relevant methods. We provide implementations for the models listed in Tab. 2.

To run an algorithm, we start by instantiating a model with corresponding hyperparameters, e.g. `model = BSC_ET(D, H, Hprime, gamma)` were `D` and `H` are observed and latent dimensions respectively and `Hprime` and `gamma` approximation parameters. We proceed by initializing the annealing class, e.g. `anneal = LinearAnnealing(150)`. Using a dictionary to store the data under the key `'y'` we call the `standard_init` method to randomly initialize the parameters, e.g. `params = model.standard_init({"y":data})`. The `EM` class is initialized as `em = EM(model=model, anneal=anneal)` and we train the model with `em.run()`. We can then apply the `inference` method to extract approximate posterior information about the data, e.g. `res=model.inference(anneal,em.lparams,data)`. This yields, e.g. the (estimated) most likely latent variable configurations (`res["s"]`) and corresponding approximate posterior probabilities (`res["p"]`) for each data point as well as additional information specific for each model.

4. Related Software Libraries

Most libraries for sparse coding or dictionary learning are based on deterministic objectives: The SPAMS library (Mairal et al., 2010, 2009) contains a collection of deterministic sparse coding algorithms with L_1, L_2 and L_∞ regularization (C++ based, interfaces to Matlab, R and Python). Similarly `MLPack` (Curtin et al., 2018) contains standard L_1/L_2 regularized sparse coding. Scikit-learn (Pedregosa et al., 2011) also contains a standard (deterministic and L_1 regularized) sparse coding version and provides standard NMF or LDA implementations. `Sparsenet` and `glm-ie` both use a continuous and probabilistic sparse coding data model, and both require the data model to provide monomodal a-posterior distributions for convex optimization. The data models used by `Sparsenet` or `glm-ie` consequently do not overlap nor do the parameter optimization methods. Sparsenet implements the original algorithm by Olshausen and Field (1996) based on MAP, and `glm-ie` (Nickisch, 2012) provides sophisticated inference for the generalized (sparse) linear model. `libDAI` (Mooij, 2010) and `Libra` (Lowd and Rooshenas, 2015) are libraries for inference and learning in general graphical models. None of the libraries is as optimized for probabilistic sparse coding as `ProSper` or provide its efficient variational EM approach. But `libDAI` and `Libra` are much more general in the graphical data models that can be treated. While `libDAI` focuses more on inference, `Libra` focuses more on learning and structure learning.

Acknowledgments

We would like to thank Julian Eggert for the many discussions. We would also like to acknowledge support by the DFG in grants SFB 1330 (HAPPAA, ID 352015383) and EXC 2177/1 (Hearing4A, ID 390895286). Early stages of the work were funded by the DFG (grant LU 1196/4-2) and the BMBF (grant 01GQ0840).

References

Jörg Bornschein, Marc Henniges, and Jörg Lücke. Are V1 receptive fields shaped by low-level visual occlusions? A comparative study. *PLOS Computational Biology*, 9(6):e1003062, 2013.

Ryan R. Curtin, Marcus Edel, Mikhail Lozhnikov, Yannis Mentekidis, Sumedh Ghaisas, and Shangtong Zhang. mlpack 3: a fast, flexible machine learning library. *Journal of Open Source Software*, 3:726, 2018. doi: 10.21105/joss.00726. URL `https://doi.org/10.21105/joss.00726`.

Zhenwen Dai, Georgios Exarchakis, and Jörg Lücke. What are the invariant occlusive components of image patches? a probabilistic generative approach. In *Advances in Neural Information Processing Systems*, pages 243–251, 2013.

G. Exarchakis, M. Henniges, J. Eggert, and J. Lücke. Ternary sparse coding. In *Proceedings LVA/ICA*, LNCS. Springer, 2012. in press.

Georgios Exarchakis and Jörg Lücke. Discrete sparse coding. *Neural computation*, 29(11): 2979–3013, 2017.

Enrico Guiraud, Jakob Drefs, and Jörg Lücke. Evolutionary expectation maximization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 442–449, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5618-3. doi: 10.1145/3205455.3205588. URL `http://doi.acm.org/10.1145/3205455.3205588`.

M. Henniges, G. Puertas, J. Bornschein, J. Eggert, and J. Lücke. Binary sparse coding. In *Proceedings LVA/ICA*, LNCS 6365, pages 450–57. Springer, 2010.

Daniel Lowd and Amirmohammad Rooshenas. The libra toolkit for probabilistic models. *Journal of Machine Learning Research*, 16:2459–2463, 2015.

J. Lücke and J. Eggert. Expectation truncation and the benefits of preselection in training generative models. *JMLR*, 11:2855–900, 2010.

J. Lücke and M. Sahani. Maximal causes for non-linear component extraction. *JMLR*, 9: 1227–67, 2008.

J. Lücke and A.-S. Sheikh. Closed-form EM for sparse coding and its application to source separation. In *LVA/ICA*, LNCS, pages 213–221. Springer, 2012.

J Mairal, F Bach, J Ponce, and G Sapiro. Online learning for matrix factorization and sparse coding. *JMLR*, 11, 2010. URL `http://portal.acm.org/citation.cfm?id=1756008`.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *ICML*, page 87, 2009.

JM. Mooij. libdai: A free and open source c++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11:2169–2173, August 2010.

H. Nickisch. glm-ie: The generalised linear models inference and estimation toolbox. *Journal of Machine Learning Research*, 13:1699–1703, May 2012.

B. Olshausen and D. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–9, 1996.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

G. Puertas, J. Bornschein, and J. Lücke. The maximal causes of natural scenes are edge filters. In *Advances in Neural Information Processing Systems*, volume 23, pages 1939–1947. 2010.

Abdul-Saboor Sheikh, Jacquelyn A. Shelton, and Jörg Lücke. A truncated em approach for spike-and-slab sparse coding. *JMLR*, 15:2653–2687, 2014.

Abdul-Saboor Sheikh, Nicol S. Harper, Jakob Drefs, Yosef Singer, Zhenwen Dai, Richard E. Turner, and Jörg Lücke. Strfs in primary auditory cortex emerge from masking-based statistics of natural sounds. *PLOS Computational Biology*, 15(1):e1006595, 01 2019. doi: 10.1371/journal.pcbi.1006595. URL `https://doi.org/10.1371/journal.pcbi.1006595`.