Approximate EM Learning on Large Computer Clusters

Jörg Bornschein[†], **Zhenwhen Dai, and Jörg Lücke**
Frankfurt Institute for Advanced Studies, Goethe-University Frankfurt, Germany
[†]Corresponding author: bornschein@fias.uni-frankfurt.de

An important challenge in the field of unsupervised learning is not only the development of algorithms that infer model parameters given some dataset but also to implement them in a way so that they can be applied to problems of realistic size and to sufficiently complex benchmark problems. We developed a lightweight, easy to use MPI (Massage Passing Interface) based Python framework that can be used to parallelize a variety of Expectation Maximization (EM) based algorithms. We used this infrastructure to implement standard algorithms such as Mixtures of Gaussians (e.g., [1]), Sparse Coding [2], or probabilistic PCA [3, 4], as well as novel algorithms such as Maximal Causes Analysis [5, 6], Occlusive Causes Analysis [7], Binary Sparse Coding [8] or mixture models for visual object learning [9, 10]. Once integrated into the framework the algorithms can be executed on large numbers of processor cores and can be applied to large sets of data. Some of the numerical experiments we performed ran on InfiniBand interconnected clusters and used up to 4000 parallel processor cores with more than 10^{17} floating point operations. Current experiments on a new cluster use still more cores (Loewe CSC, >10 000 cores). For reasonably balanced meta-parameters (number of data points vs. number of latent variables vs. number of model parameters to be inferred), we observe close to linear runtime scaling behavior with respect to the number of cores in use. Furthermore, we use algorithms running on GPU equipped compute nodes. The implementations use MPI to parallelize across the GPUs of the distributed memory cluster, and use OpenCL to perform high-throughput computation on locally stored data (algorithms are executed on the Scout cluster, >100 GPUs, see Fig. 1).

Exact and approximate EM learning. The algorithms we implemented are all based on probabilistic generative models (see, e.g., [11, 12] for an overview). We will briefly describe the general approach and common methods for parameter optimization. To illustrate parallelized optimization, we discuss the implementation and challenges of a specific model and training scheme.

Given a generative model, one of the most frequently used criteria for parameter optimization is the maximization of the data likelihood under the generative model: $\mathcal{L}(\Theta) = p(\vec{y}^{(1)}, \ldots, \vec{y}^{(N)} \,|\, \Theta)$, where $\vec{y}^{(n)}$ is a data point. A standard approach to maximize the likelihood is the EM algorithm. EM is a gradient approach that iteratively updates the parameters Θ of the model. Each iteration consists of an E-step and an M-step. For algorithms such as Mixture of Gaussians approaches or probabilistic PCA, the E- and M-steps are computationally tractable. However, for almost all models that go beyond such elementary approaches, the E-steps become computationally intractable. Typical examples of generative models with intractable E-steps are multiple-causes models, i.e., models that are used to analyse the compositional structure of data. Among the most frequently used such models are variants of Sparse Coding. The complexity of exact EM learning for such models typically scales exponentially with the number of data components (usually all combinations of all components have to be evaluated).

The fundamental problem of intractable learning for most generative models have motivated extensive research on tractable approximations to exact EM. Prominent approaches are maximum a posteriori (MAP) approximations (e.g., [2]), variational EM (e.g., [13]), or Expectation Propagation (EP; [14]). Such approaches use analytical approximations to exact E-step solutions. Another class of approaches are sampling approaches that include importance sampling, Gibbs sampling, or Markov Chain Monte Carlo (see, e.g., [11, 12] for overviews). These approaches seek efficient ways to draw samples from complex posterior distributions to approximate exact EM. Approximate EM approaches differ in their accuracy, computational cost, and in their suitability for parallelization.

In the development of novel learning algorithms in our group we work with exact EM for mixture models and probabilistic PCA, and with variational EM and sampling approaches for multiple-causes models. The algorithms listed in the beginning have been implemented and parallelized to run on large-scale clusters. As an example to discuss the challenges for parallelization we will here use a variant of Sparse Coding and a variational EM approach for learning. Much of the learning algorithm's structure will be similar to that of other models.

Parallelization of Sparse Coding variants. The probabilistic generative formulation of Sparse Coding (SC) consists of a sparse prior distribution (few hidden units with values significantly different from zero), and a Gaussian noise distribution with $W\vec{s}$ as mean value. We focus on a SC variant with binary hidden variables but for completeness we also state a (classical) continuous version:

$$\begin{array}{lcl@{\qquad\qquad}lcl}
\multicolumn{3}{c}{\textbf{Continuous Sparse Coding}} & \multicolumn{3}{c}{\textbf{Binary Sparse Coding}} \\
p(\vec{s}|\Theta) & = & \displaystyle\prod_{h=1}^{H} \frac{1}{\pi\,(1+s_h^2)} & p(\vec{s}|\Theta) & = & \displaystyle\prod_{h=1}^{H} \lambda^{s_h}\big(1-\lambda\big)^{1-s_h} \\
p(\vec{y}\,|\,\vec{s},\Theta) & = & \mathcal{N}(\vec{y};\,W\vec{s},\sigma^2\mathbb{1}) & p(\vec{y}\,|\,\vec{s},\Theta) & = & \mathcal{N}(\vec{y};\,W\vec{s},\sigma^2\mathbb{1})
\end{array} \tag{1}$$

where $W \in \mathbb{R}^{D\times H}$ and H denotes the number of hidden variables s_h. Θ is the set of parameters given by $\Theta = (W,\sigma)$.

Independent of the prior, the parameter update rules (M-step equations) are given by:

$$W^{\mathrm{new}} = \Big(\sum_{n\in\mathcal{M}} \vec{y}^{\,(n)}\,\langle\vec{s}\rangle_{q_n}^{T}\Big)\Big(\sum_{\tilde{n}\in\mathcal{M}} \big\langle\vec{s}\,\vec{s}^{\,T}\big\rangle_{q_{\tilde{n}}}\Big)^{-1},\ \ \sigma^{\mathrm{new}} = \sqrt{\frac{1}{|\mathcal{M}|\;D}\sum_{n\in\mathcal{M}}\Big\langle\big|\big|\vec{y}^{\,(n)} - W\,\vec{s}\,\big|\big|^2\Big\rangle_{q_n}}$$

$$\text{where}\qquad \langle g(\vec{s})\rangle_{q_n} = \sum_{\vec{s}} q_n(\vec{s};\Theta)g(\vec{s}) \qquad \text{for a function}\ \ g(\vec{s}). \tag{2}$$

The set $\mathcal{M}$ contains all or a subset of all data points. For the continuous case the sum in (2) has to be replaced by an integral. Note that M-step equations of probabilistic PCA or Factor Analysis are essentially of the same form.

The expectation values (E-step equations) computed in (2) are intractable for large numbers of hidden variables and have to be approximated for large-scale applications. Variational EM approximations replace the optimal choice $q_n(\vec{s};\Theta) = p(\vec{s}\,|\,\vec{y}^{\,(n)},\Theta)$ by distributions which result in tractable evaluations of expectation values (2). We consider three cases, exact EM, Expectation Truncation (ET; [15]), and maximum a posteriori (MAP) estimation. The latter two can both be regarded as instances of variational EM:

$$\begin{array}{rlcll}
\textbf{Exact} & q_n(\vec{s};\Theta) & = & p(\vec{s}\,|\,\vec{y}^{\,(n)},\Theta) & \\
\textbf{ET} & q_n(\vec{s};\Theta) & = & \frac{1}{A}\,p(\vec{s}\,|\,\vec{y}^{\,(n)},\Theta)\,\delta(\vec{s}\in\mathcal{K}_n), & \displaystyle\sum_{\vec{s}}\delta(\vec{s}\in S)f(\vec{s}) \;=\; \sum_{\vec{s}\in S} f(\vec{s}) \\
\textbf{MAP} & q_n(\vec{s};\Theta) & = & \delta(\vec{s}-\vec{s}^{\,\max}), & \displaystyle\sum_{\vec{s}}\delta(\vec{s}-\vec{s}^{\,\max})f(\vec{s}) \;=\; f(\vec{s}^{\,\max})
\end{array} \tag{3}$$

For ET, A is a normalization constant. Note that $\delta(\vec{s}\in S)$ is one if $\vec{s}\in S$ and zero otherwise, and that sums have to be replaced by integrals in the continuous case. The set $\mathcal{K}_n$ for the ET approximation is chosen to contain most of the posterior mass with high probability (see [15]). For the MAP approximation $\vec{s}^{\,\max}$ is chosen to be the maximum of the posterior distribution $p(\vec{s}\,|\,\vec{y}^{\,(n)},\Theta)$ (see, e.g., [2]).

We will consider training using ET which represents a compromise between exact EM and the relatively severe MAP estimation. The update equations for each of the model parameters usually contain sums over the expectation values $\langle g(\vec{s})\rangle_{q_n}$, where the index n in (2) may run over a subset of data points (compare [15]). To parallelize an EM iteration we compute the expectation values $\langle g(\vec{s})\rangle_{q_n}$ independently for each data point and integrate the results in a final step by using a collective sum-reduction. Note that the parameter update equations for some model parameters contain two or more of these sums (e.g. the W-update in

Computer Clusters		
Name	# CPU cores	# GPUs
FIAS local	64	6
GPU-Scout	144	108
Fuchs CSC	∼4500	0
Loewe CSC	∼19000	786

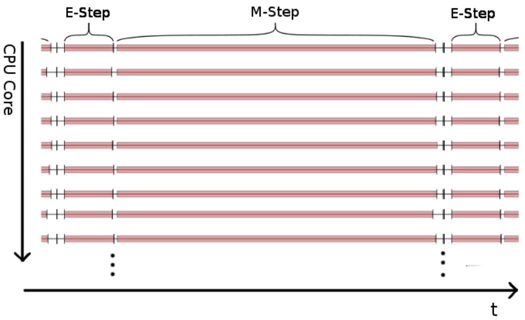

Figure 1: **Left**: The clusters we used for our experiments. **Right**: A typical runtime trace. Red sections indicate that a processor core was occupied with computation. Gaps indicate that a core was either waiting for or performing communication via MPI.

(2)). We therefore split the training dataset into partitions of approximately equal size and distribute these to the compute nodes. To perform a global parameter update, each node performs a partial sum over the locally computed expectation values $\langle g(\vec{s})\rangle_{q_n}$ and finally participates in a global reduction operation. In case of the W-update in (2) we need to perform a global reduction over a matrix of type $\mathbb{R}^{H\times D}$ for the first multiplier, and second global reduction over a matrix of type $\mathbb{R}^{H\times H}$ for the second multiplier. In the case of the σ-update, the global reduction only covers a single scalar value[1]. This illustrates, that the communication cost for a global parameter update is relatively small compared to the computation that has to be performed. Note that to evaluate the expectation values $\langle g(\vec{s})\rangle_{q_n}$, all the approximate posterior probabilities $q_n(\vec{s},\Theta)$ for all the data points $\vec{y}^{(n)}$ and for all hidden states $\vec{s}\in\mathcal{K}_n$ that are of interest have to be computed.

Per core performance and vectorization. Using MPI and Python results in a convenient environment to run large-scale machine learning experiments. Although Python is an interpreted and relatively slow language (in terms of instructions per second), its execution speed is not the limiting factor: The evaluation of the approximate posterior probabilities $q_n(\vec{s},\Theta)$ in high dimensional parameter and data spaces is one of the computationally intensive steps of a typical component extraction algorithm such as Sparse Coding. For all the models we implemented so far, the required expectation values can be computed through vectorized notations. These vectorized expressions are evaluated efficiently by Python. Often it is also possible (and convenient) to aggregate the evaluation of multiple hypotheses $\vec{s}\in\mathcal{K}_n$ into vectorized expressions. Applying this kind of vectorization merges a large number of mathematical operations into a single Python expression. As a result, the Python expressions in the inner loops of our algorithm typically involve thousands of mathematical operations. These observations are backed by profiling runs: In all our experiments the time spent in the Python interpreter was less than 10% of the total runtime, typically even less than 5%.

Looking closer at the runtime behaviour of our implementation, it became evident that the evaluation of exponential and logarithm functions often consumed a significant fraction of the computing time. Highly optimized implementations of these transcendental functions can be found in the AMD Core Math Library (ACML) and in the Intel Math Kernel Library (MKL) and provided a significant speedup compared to the standard lib-c implementation

Software architecture. We designed the software framework to be lightweight and to facilitate the development of parallel EM learning programs. The basic idea behind the software architecture is to encapsulate model dependent functionality into stateless objects which provide a set of methods like `Estep`, `Mstep`, `generate_data` etc. Although model classes are stateless with respect to model parameters, data points and annealing parameters, they might store meta-parameters that do not change over the course of a full EM training run. An instance of an `EM` class handles the outer iterative EM loop and executes an EM iteration (and potentially provides a predefined annealing scheme). The `EM` class ensures that the EM iteration functions receives the current set of model parameters, a local partition of the data points and all other necessary parameters as arguments.

[1] Similarly, note that ET also allows for the update of the prior parameter λ [8].

The framework itself is currently used by a group of eight users and is developed using typical development tools such as version control, unity tests etc.

Conclusion and future challenges. The parallelization strategy we have chosen demonstrated good scaling behavior for the models we implemented and for the compute clusters we used. An interesting observation is, that we usually only require a small subset of the functionality provided by MPI: Reductions for the parameter updates, broadcasts for (e.g.) parameter noise, and for ET based algorithms, a collective selection of data points across all nodes. This observation and the fact that the compute nodes mostly operate on their locally stored data subsets indicate that our overall program structure could be translated to so-called map-reduce based programming model ([16], not to be confused with the MAP approximation scheme). By using a map-reduce infrastructure like, e.g., Hadoop [17], we can in future work make further advances towards automatically distributed storage and, potentially, a higher aggregate throughput by running multiple EM jobs in parallel on the same set of nodes. Most importantly, we can gain better resilience to node failures—which becomes an increasingly important aspect for massively parallel programs. So far this has been mitigated by saving a checkpoint containing the full set of the learned parameters at the end of each EM iteration. In combination with improvements on the analytical side, e.g. by using optimized batch sizes with partial EM, the current scope of applications can thus be extended still further.

Acknowledgements. We acknowledge funding by the projects DFG LU 1196/4-1 and BMBF 01GQ0840 and support by the Frankfurt Center for Scientific Computing (CSC Frankfurt).

References

[1] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39:1–38, 1977.

[2] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607 – 609, 1996.

[3] S. Roweis. EM algorithms for PCA and SPCA. *NIPS*, pages 626–632, 1998.

[4] M. E. Tipping and C. M. Bishop. Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 61(3), 1999.

[5] J. Lücke and M. Sahani. Maximal causes for non-linear component extraction. *Journal of Machine Learning Research*, 9:1227 – 1267, 2008.

[6] G. Puertas, J. Bornschein, and J. Lücke. The Maximal Causes of Natural Scenes are Edge Filters. In *NIPS 23, in press.*, 2010.

[7] J. Lücke, R. Turner, M. Sahani, and M. Henniges. Occlusive Components Analysis. In *NIPS 22*, pages 1069–1077, 2009.

[8] M. Henniges, G. Puertas, J. Bornschein, J. Eggert, and J. Lücke. Binary Sparse Coding. In *Proc. LVA/ICA*, volume 6365 of *LNCS*, pages 450–457. Springer, 2010.

[9] Z. Dai and J. Lücke. A probabilistic generative approach to invariant visual inference and learning. In *Proc. BCCN*, volume 4, 2010.

[10] B. J. Frey and N. Jojic. Transformation-invariant clustering using the EM algorithm. *IEEE Pattern Analysis and Machine Intelligence*, pages 1–17, 2003.

[11] P. Dayan and L. F. Abbott. *Theoretical Neuroscience.* MIT Press, Cambridge, 2001.

[12] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[13] M.I. Jordan. *Learning in graphical models.* Kluwer Academic Publishers, 1998.

[14] T. P. Minka. Expectation propagation for approximate Bayesian inference. In *Porc. UAI*, pages 362–369, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[15] J. Lücke and J. Eggert. Expectation truncation and the benefits of preselection in training generative models. *Journal of Machine Learning Research*, 11:2855 – 2900, 2010.

[16] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Operating Systems Design and Implementation*, pages 137–149, 2004.

[17] A Bialecki, M. Cafarella, D. Cutting, and O. O'Malley. Hadoop: A Framework for Running Applications on Large Clusters Built of Commodity Hardware, 2005. http://lucene.apache.org/hadoop/.