Approximate EM Learning on Large Computer Clusters

Jörg Bornschein, Zhenwhen Dai and Jörg Lücke
Frankfurt Institute for Advanced Studies, Goethe-University, Germany

Summary

- parallel implementation of Expectation Maximization (EM) [1] based algorithms for large data sets.
- parallelization based on MPI
- running experiments on up to 5000 cores in parallel [2]
- lightweight and easy to use
- framework implemented in Python
- supporting GPGPU accelerated computation using PyCUDA and PyOpenCL
- applicable to a variety of algorithms. Currently implemented: Mixture of Gaussians, Sparse Coding, Binary Sparse Coding, Maximal Causes Analysis

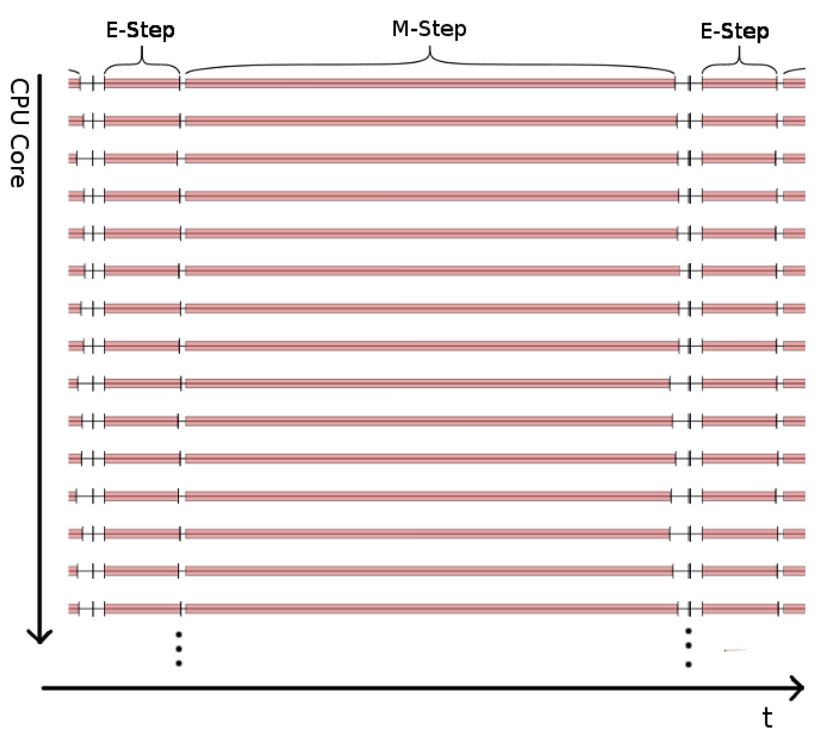
Parallelization strategy

- partition according to data points
- compute sufficient statistics on local set of data points
- use (sum-)reductions to aggregate statistics in M-step
- if necessary use global operation to select data points (e.g.: sort data points according to their posterior probability when using ET)

Computer Clusters:

Name	# CPU cores	# GPUs
GPU-Scout	144	108
FIAS	500	12
Fuchs CSC	∼4500	0
Loewe CSC	∼19000	786

Typical runtime trace:



Multiple cause model with Expectation Truncation (ET)

Binary Sparse Coding generative model:

$$p(\vec{s}\,|\,\Theta) \;=\; \prod_{h=1}^{H} \lambda^{s_h}(1-\lambda)^{1-s_h}$$

$$p(\vec{y}\,|\,\vec{s},\Theta) \;=\; \mathcal{N}(\vec{y};\, W\vec{s}, \sigma^2)$$

where

$\vec{y} \in \mathbb{R}^D$	observed variables	λ	prior parameter
$\vec{s} \in \{0,1\}^H$	hidden variables	σ	noise level
$W \in \mathbb{R}^{D\times H}$	basis functions		

The parameter update equations (M-step) are given by:

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

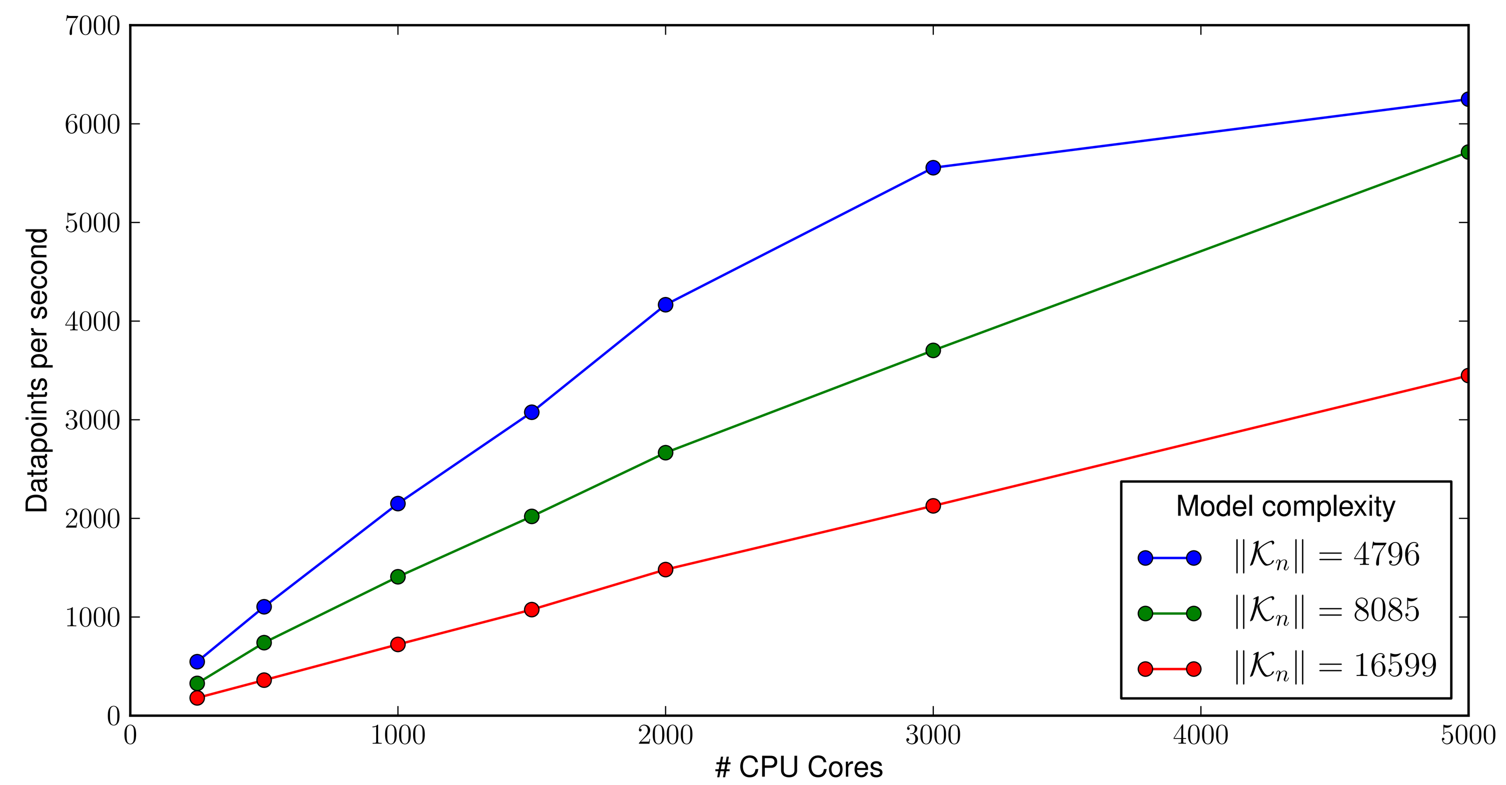
$$\lambda^{\mathrm{new}} = A(\lambda)\,\frac{1}{|\mathcal{M}|}\sum_{n\in\mathcal{M}} \langle|\vec{s}\,|\rangle_{q_n}$$

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

with $\langle g(\vec{s})\rangle_{q_n} = \sum_{\vec{s}} q_n(\vec{s};\Theta)g(\vec{s})$ for a function $g(\vec{s})$ and

$q_n(\vec{s};\Theta) = p(\vec{s}\,|\,\vec{y}^{(n)},\Theta)$ (exact EM) or

$q_n(\vec{s};\Theta) = \dfrac{1}{B}\,p(\vec{s}\,|\,\vec{y}^{(n)},\Theta)\,\delta(\vec{s}\in\mathcal{K}_n)$ (ET approximation [3])



Software architecture

- model specific code is encapsulated in `Model`-classes
- `Model` classes are stateless in respect to model parameters, annealing parameters and data points:

```
class SparseCoding(Model):
  def generate_data(self, model_params, N):
    ...

  def EM_step(self, model_params, annealing_params, my_data):
    ...
```

- `EM` class handles data and drives computation:

```
class EM:
   def set_data(self, data):
     ...

   def set_annealing(self, annealing):
     ...

   def set_model_params(self, model_params):
     ...

   def run(self):
     ...
```

- `Annealing` objects determine variables that parameterize the annealing scheme (e.g. temperature)
- Utility functions: data input/output, runtime tracing, etc.

Conclusions

- parallelization of many EM based algorithms is straight forward
- a framework providing infrastructure (input/output, data-handling, etc.) is neccessary to facilitate parallel implementations
- using MPI and Python results in a convenient environment to run large-scale machine learning experiments.
- implementation demonstrates good scaling properties

Sourcecode will be available at

`http://fias.uni-frankfurt.de/~bornschein`

References

[1] R.Neal, G. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants *Learning in Graphical Models* 355-368, 1998
[2] G. Puertas, J. Bornschein and J. Lücke. The Maximal Causes of Natural Scenes are Edge Filters. *NIPS* 23:1939-1947, 2010
[3] J. Lücke, J. Eggert. Expectation Truncation and the Benefits of Preselection in Training Generative Model. *JMLR* 11:2855–2900, 2010

This project was supported by the German Federal Ministry of Education and Research (BMBF) within the "Bernstein Focus: Neurotechnology Frankfurt" through research grant 01GQ0840 and by the German Research Foundation (DFG) in the project LU 1196/4-1.