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:

<table>
<thead>
<tr>
<th>Name</th>
<th># CPU cores</th>
<th># GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU-Scout</td>
<td>144</td>
<td>108</td>
</tr>
<tr>
<td>FIAS</td>
<td>500</td>
<td>12</td>
</tr>
<tr>
<td>Fuchs CSC</td>
<td>~4500</td>
<td>0</td>
</tr>
<tr>
<td>Loewe CSC</td>
<td>~19000</td>
<td>786</td>
</tr>
</tbody>
</table>

Typical runtime trace:

Multiple cause model with Expectation Truncation (ET)

Binary Sparse Coding generative model:

\[
p(\mathbf{s}|\Theta) = \prod_{j=1}^{K} (1 - \lambda_j)^{1 - \mathbf{q}_j} \nu_{\mathbf{q}_j}
\]

\[
p(\mathbf{y}|\mathbf{s}, \Theta) = \mathcal{N}(\mathbf{y}; \mathbf{W} \mathbf{s}, \sigma^2)
\]

where $\mathbf{y} \in \mathbb{R}^D$ observed variables, $\mathbf{s} \in (0, 1)^H$ hidden variables, $\lambda$ prior parameter, $\sigma$ noise level. $W \in \mathbb{R}^{D \times M}$ basis functions

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

\[
W^{\text{new}} = \left( \sum_{n,M} \langle g(\mathbf{s}) \rangle_\Theta \right)^{-1} \left( \sum_{n,M} \langle \mathbf{y} g(\mathbf{s}) \rangle_\Theta \right)
\]

\[
\lambda^{\text{new}} = A(\lambda) \frac{1}{|M|} \sum_{n,M} \langle \mathbf{y} \delta \rangle_\Theta
\]

\[
\sigma^{\text{new}} = \frac{1}{\sqrt{|M|}} \sum_{n,M} \left( \| \mathbf{y} - W \mathbf{s} \|_2^2 \right)_\Theta
\]

with $\langle g(\mathbf{s}) \rangle_\Theta = \sum_{n,M} q_n(\mathbf{s}|\Theta) g(\mathbf{s})$ for a function $g(\mathbf{s})$ and

\[
q_n(\mathbf{s}|\Theta) = \frac{p(\mathbf{s}|\mathbf{y}, \Theta)}{\sum_{\mathbf{s}} p(\mathbf{s}|\mathbf{y}, \Theta)} \quad \text{(exact EM)}
\]

\[
q_n(\mathbf{s}|\Theta) = \frac{1}{B} q_n(\mathbf{s}|\mathbf{y}, \Theta) (\mathbf{s} \in \mathcal{K}_\Theta) \quad \text{(ET approximation [3])}
\]

Software architecture
- model specific code is encapsulated in Model-classes
- Model classes are stateless in respect to model parameters, annealing parameters and data points:
  ```python
class SparseCoding(Model):
    def generate_data(self, model_params, N):
      ...
    def EM_step(self, model_params, annealing_params, my_data):
      ...
  
  class EM:
    def set_data(self, data):
      ...
    def set_annealing(self, annealing):
      ...
    def set_model_params(self, model_params):
      ...
    def run(self):
      ...
```
- EM class handles data and drives computation:
- 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 straightforward
- a framework providing infrastructure (input/output, data-handling, etc.) is necessary 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

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.