Programming Project: Restricted Boltzmann Machine

Project Definition

Work in groups of two students. Exceptions have to be arranged with the instructor.

1. Implement a generic RBM over binary random variables. The number of visible and hidden units should be settable at run time. Implement methods for training (using Stochastic Maximum Likelihood a.k.a. Persistent Contrastive Divergence) and for sampling (visible) instances from it.

2. Train your RBM on MNIST[1] data, and draw random images from your trained RBM. Randomly-drawn images should represent at least 3 different digits. Show at least 10 randomly-drawn images in your report.

   Note that you need to binarize the gray-scale MNIST images first.

3. Systematically explore and discuss the effect on *convergence* behavior during training and qualitative *performance* of the trained RBM of the number of hidden units and of at least one more parameter including:

   - the learning rate α
   - the number of Markov chains
   - the length of the burn-in phase of the Gibbs sampler
   - the number of instances dropped between retained instances during Gibbs sampling

   If your group counts three students, systematically investigate and discuss the influence of two more parameters.

Implement everything in terms of generic math. Do not use any 3rd-party code, libraries, frameworks, etc. specific to graphical models, machine learning, etc.

Hints

- The (smaller) “test set” of the MNIST data should be sufficient for training.

  This Python snippet is one way to load it:

```
import numpy as np

dt = np.dtype('>u4, >u4, >u4, >u4, (10000,784)u1')
mnist = np.fromfile('t10k-images-idx3-ubyte', dtype=dt)['f4'][0].T
imgs = np.zeros((784, 10000), dtype=np.dtype('b'))
imgs[mnist > 127] = 1
```

- If you work in a scripting language (which is recommended), it is (as always) crucial that you vectorize your code to avoid explicitly iterating over units; otherwise your code will probably run very slowly.

- Before embarking on the MNIST data, validate your RBM code using the simple example of Exercise 2 from the lecture notes.

Implementation

You are free to choose almost any programming language. Since this project involves mostly maths and graphics, you are advised to implement your solution in an appropriate, high-level scripting language such as R[2], Python SciTools[3], Octave[4][5], or similar. Non-free programming languages (Matlab) are discouraged. Your code needs to be able to run on the instructor’s Linux machine without installing substantial extra software. If in doubt, check with the instructor.

Your program should start from a command line (shell, R or Octave interpreter), should not accept any required parameters on the command line, and and should not read anything from standard input.

[1] http://yann.lecun.com/exdb/mnist/
[2] https://www.r-project.org/
[3] https://pypi.org/project/SciTools/
[4] http://www.gnu.org/software/octave/
[5] a free Matlab clone

Submission

Prepare a report of a few pages with the following content:

- Give the names of all members of the group. If the members' contributions differ substantially, explain their respective contributions.
- Discuss any algorithmic choices you made. Explain any deviations from the methods discussed in class.
- Give illustrations, numbers and results as described above.
- Discuss your results and any interesting insights and observations.

One member of the group should submit the report in PDF format as well as the complete source code as indicated on the course Web page.