Report on Challenge for summer internship at DiCarlo Lab, MIT: Convolutional Deep Belief Networks in Python

Mojtaba Solgi
solgi@cse.msu.edu

February 18, 2011

1 Problem Declaration

The goal was to re-implement the Convolutional Deep Belief Nets by Lee et. al. (2009) [1] in Python, numpy/scipy, following PEP8 [3]. The implementation was based on a former partial implementation of the network in Matlab, available from [4], and also an older article by Lee et al. (2008) [2] which explained some details of the algorithms.

2 The Solution

The network laid out in [1] was implemented only in its unsupervised mode. However, it can be easily extended to learn classification/regression by initializing the weights of a traditional back-propagation MLP using the weights learned in this network.

2.1 Input

The network was tested on natural images provided by the original Matlab implementation [4], as well as with a face expression image database collected by the author [5] (submitted with the solution).

Images were preprocessed by filtering them with a zero-phase whitening/lowpass filter $R(f) = fe^{-(f/f_0)^4}$ where $f_0 = 0.4 \times (w + h)/2$ where h and w are the height and width of the image, respectively.

2.2 Design

An object oriented design was employed for the project. This significantly improves the readability and scalability of the code compared with the Matlab implementation.

A *layered architecture* was used in the design of the project meaning that the control component of the program (the CDBN algorithm) was implemented in such a way that it is independent of the data component, and vice versa.

Below is the list of the main classes used in the project, in a top-down order:

1. **class Network:** this class encompasses the Convolutional Deep Belief Network (CDBN). It maintains a list of layers, and the index of the layer currently under training.

2. **class Layer:** defines one layer of the multiple layers in the network. Its most important data members include a list of bases (also called "groups" in [1]), its input data array (also called positive data in the literature), its belief data array (also called negative data in the literature) and a link to the previous layer in the network.

3. **class Base:** is a "group" of units in the layer that share the same weight vector, and convolve over the input array to compute an output array.

3 Setup and Run

1. Place the dataset (only image is currently supported) to a directory (the program was tested on natural images dataset, in *./data/natural_imgs*, and face images, in *./data/face_imgs*)

2. Set the parameters of the model and data in parameter to your desired values. Two sample parameter files are provided for the natural image and face datasets (*params_natural_imgs.py*, and *params_face_imgs.py*) respectively.

3. Set the value of *params_filename* in *config.py* to point to your parameters file[1].

4. To run the program type either:

   (a) *python main.py* for faster, terminal mode

   (b) *python GUI.py* for user interface mode

   In order to run the program, you need to have write permission to the results directory (set via in $misc\_params['results\_dir']$ parameters module) where the programs saves the outputs of the simulation.

4 Results

The network was tested for unsupervised, hierarchical feature detection on two datasets; natural scene images and face images. On natural images, it managed to replicate the result in [1] by developing orientation feature detectors in the first layer (Fig. 1).

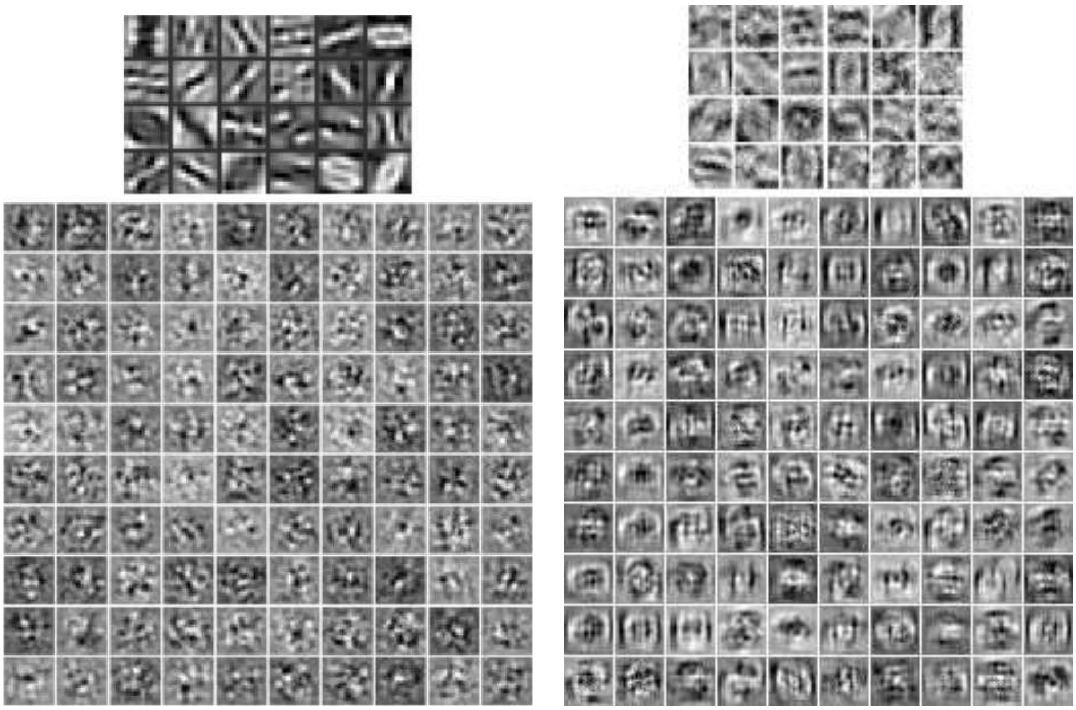
Applying the network on face images resulted in developing lower level face feature (e.g. noses, eyes, etc.) in the first. This result is also consistent with [1] (Fig. 1).

As shown in Fig. 1, the implementation did not produce results similar to [1] in the second layers. It seems that replicating their exact results for higher layers would require careful selection of many parameters in the model. I emailed the authors and asked them for the parameter set they used to get those results, but did not hear back from them.

[1] Alternatively, you could pass the parameters file names via command-line argument (i.e. *python main.py <params_file.py>*).

To ensure the network is improving its internal representation during learning, I plotted the error made by the network in different epochs (Fig. 2).

Fig. 3 shows samples of negative data sampled from the generative model of the network, in comparison their positive data (ground truth).



(a) Natural images (b) Face images

Figure 1: Filters developed after training network on **(a)** natural images and **(b)** face images. Boxes on **top** show 24 filters corresponding to 24 bases in the first layer of the network, and boxes at the **bottom** show 100 filters corresponding to 100 bases in the second layer. Filters of the second layer were visualized as a weighted linear combination of first layer filters, as suggested by [1].

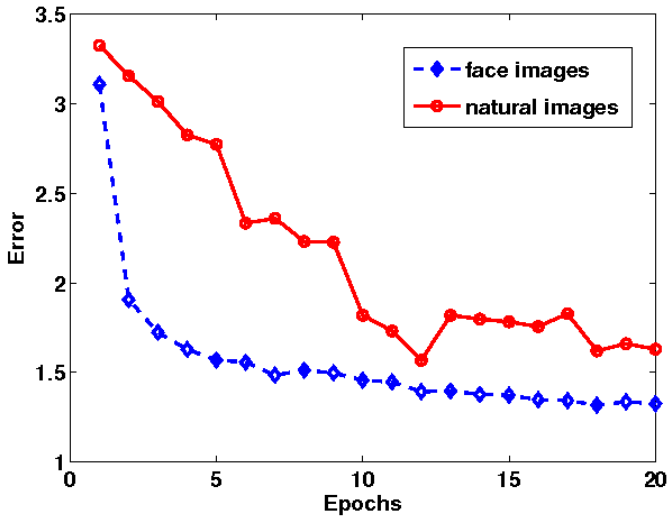
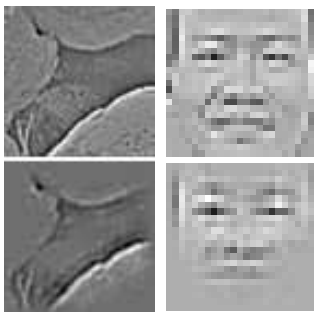


Figure 2: Amount of error made in epochs for natural scene and face images. Error was measured as the Euclidean norm of the difference between positive data (input to the network) and negative data (belief of the network). i.e. $error = ||\mathbf{pos\_data} - \mathbf{neg\_data}||$.



(a) (b)

Figure 3: Samples of recreation of input by the generative model of the network. **(a)** Sample from natural images, **(b)** sample from face images. Images on top are the input images (positive data), and bottom images show their corresponding recreation by the network (negative data or network's belief).

References

[1] "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations.", Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. In Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML), 2009.

[2] "Sparse deep belief net model for visual area V2.", Honglak Lee, Chaitu Ekanadham, and Andrew Y. Ng. Advances in Neural Information Processing Systems (NIPS) 20, 2008.

[3] http://www.python.org/dev/peps/pep-0008

[4] http://ai.stanford.edu/~hllee/softwares/code_crbm.tgz

[5] http://www.cse.msu.edu/~solgi/databases/MSU-face_expressions_data.tgz

Appendices

A List of files in the package

```
base.py
config.py
convDBN.py
GUI.py
image_data.py
__init__.py
layer.py
main.py
params_face_imgs.py
params_natural_imgs.py
utils.py
```

B Known Issue

1. The program gets quite slow when the network becomes deeper (for the default 2-layer network in $params\_face_imgs.py$, it takes over 10 seconds per training example on a 2Ghz Core Dow CPU with 2GB of memory). Using static data types of Cython seems to be an easy fix for this issue.

2. The refresh rate of the GUI (set via $misc\_params['GUI\_refresh\_timeout']$ in parameters module) is not synchronized with the actual simulation steps. One needs to adjust the timeout depending on the performance of their machine.

C GUI screenshot

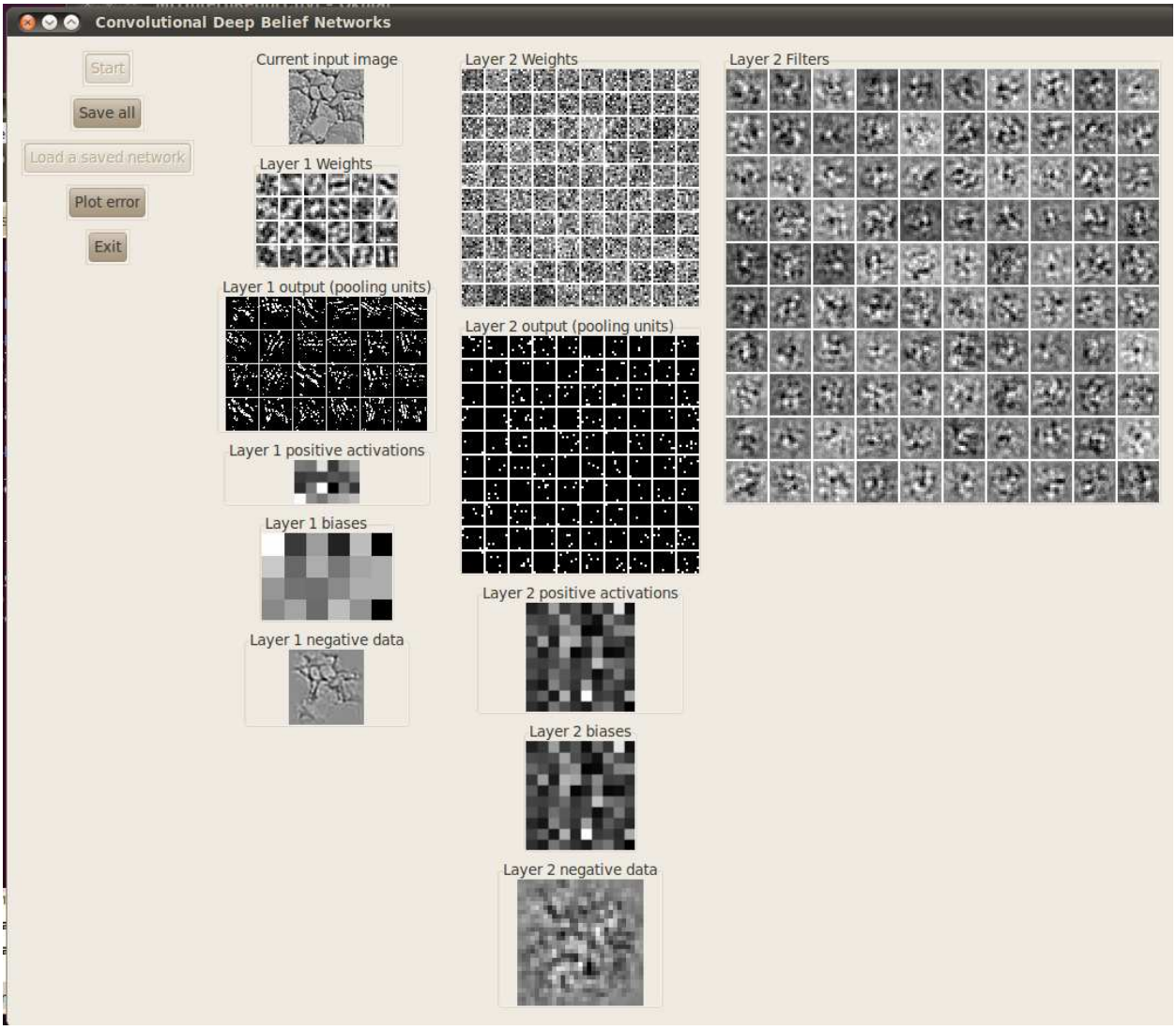


Figure 4: Sample screen-shot of the GUI developed for a 2-layer network.