Installation

Download the zip file you can find on the website, and extract it.

Linux/Mac:

- go in the mlpython subdirectory and type "make" (which will compile the necessary software for this practical session).
- go back to the root of the deep_learning directory, which contains the file "deep_learning.py" that you will be working with.

Windows:

- Install the Anaconda Python 2.7 distribution (for 64-bit Windows) available at http://continuum.io/downloads.
- Extract 'deep_learning-win64-python27.zip' and open the command line in the unzipped directory.
- In the command line, execute 'python deep_learning.py mnist_train.libsvm mnist_valid.libsvm mnist_test.libsvm <greedy_module>' to run the code.

Information about Python and Numpy

If you have never used Python, you'll find a basic tutorial here: http://docs.python.org/tutorial/ focus your attention on sections 2, 3, 4, 5 and 9; look at sections 6, 7 and 10 for more useful stuff

You will also be using the NumPy library, which provides a multidimensional array data structure (e.g. for vectors and matrices):
You will find a tutorial here: http://www.scipy.org/Tentative_NumPy_Tutorial if you are familiar-

with MATLAB, read either of these:
 * http://www.scipy.org/NumPy_for_Matlab_Users
 * http://mathesaurus.sourceforge.net/matlabnumpy.html

Details on the practical session

During this practical session, you are invited to implement two unsupervised greedy modules: RBM class, i.e. a restricted Boltzmann machine DenoisingAutoencoder class, i.e. a denoising autoencoder

An Autoencoder class is already implemented, to give you an example of what is expected. The main parts of the RBM and DenoisingAutoencoder classes are also already implemented. All that you need to fill in is the part of the "train" method that updates the parameters of the model based on each "input" (a vector) taken out of a training set. The parameters are the weight matrix "self.W" and the bias vectors "self.b" and "self.c" (the "self." suffix means that "W", "b" and "c" are fields of the object).

The RBM and DenoisingAutoencoder have a "self.learning_rate" field that corresponds to the pretraining learning rate hyperparameter to use.

RBM also has a "self.CDk" field which corresponds to the number of Gibbs sampling steps (default=1).
DenoisingAutoencoder has a "self.noise_prob" field which is the probability that should be used for stochastically corrupting each element of the input and setting it to 0 (default=0.1).

Once you have implemented these classes, you can test your implementation by running
```
python deep_learning.py mnist_train.libsvm mnist_valid.libsvm mnist_test.libsvm
<greedy_module>
```

Where mnist_{train,valid,test}.libsvm are the training, validation and test sets in libsvm format. These files are provided, and correspond to subsets of the original MNIST dataset. You can of course use any other classification data set that you have.

Also, greedy_module can either be:

none	(for no pretraining)
autoencoder	(for pretraining with the Autoencoder class)
RBM	(for pretraining with the RBM class)
denoising_autoencoder	(for pretraining with the DenoisingAutoencoder class)

The script will use the given greedy module to pretrain a 2 hidden layer classification neural network, with 200 hidden units in each layer. Before finetuning, the filters learned at the first hidden layer during pretraining will be displayed (you'll need to close the popedup window for finetuning to start).

You should get around 6.5% classification test error without pretraining, and around 5.5% error with pretraining.

Since we are using a subset of the real dataset, we should expect that results will not be as good as the ones reported in the literature, nor will the filters be as "nice looking". However, pretraining should still provide an improvement in classification performance, compared to having no pretraining.

Once you are done, feel free to play with the other hyperparameters in the script, such as the number of hidden layers, their sizes, the pretraining and finetuning learning rates, etc.