Assignment 3 : Restricted Boltzmann machines, autoencoders and deep learning

IMPORTANT : Please do not share your solution to this assignment on the web or with anyone !

In this assignment, you must implement in Python a restricted Boltzmann machine (RBM) and a denoising autoencoder, used to pre-train a neural network.

The implementation of the RBM and the autoencoder must be contained in classes named `RBM` and `Autoencoder`, that inherit from the class `Learner` of the MLPython library. The definition of the classes must be placed in files named `rbm.py` and `autoencoder.py` respectively. Thes classes support the use of the hyper-parameters :

- `lr` : learning rate of stochastic gradient descent (`float`)
- `hidden_size` : size of the hidden layer (`int`)
- `seed` : seed of the random number generator for initialization of the parameters (`int`)
- `n_epochs` : number of training iterations (`int`)

The `RBM` class must also support the following hyper-parameter :

- `CDk` : number of Gibbs step used by contrastive divergence (`int`)

On the other hand, the `Autoencoder` class must support the following hyper-parameter :

- `noise_prob` : the noise probabiliy of fixing an input to 0 (`float`)

A skeleton of the `RBM` and `Autoencoder` classes are provided in the files `rbm.py` and `autoencoder.py` available on the course's website. You only have to implement the method `train` in these files. **It is important to use the Numpy library in your implementation, so that it is efficient**.

To debug your implementations, the scripts `run_show_filters_rbm.py` and `run_show_filters_autoencoder.py` can be used to compare the learned filters (i.e. the connections of each hidden units) with those obtained by a correct implementation (see the files `rbm_filters.pdf` and `autoencoder_filters.pdf` available on the course's website).

Moreover, scripst `run_stacked_rbms_nnet.py` et `run_stacked_autoencoders_nnet.py` are available to pre-train a neural network using either the restricted Boltzmann machine or the denoising autoencoder (respectively) on the *OCR Letters* data set (the same as in the first assignment).

The script `run_stacked_rbms_nnet.py` requires the following hyper-parameters :

```
Usage: python run_stacked_rbms_nnet.py lr dc sizes pretrain_lr pretrain_n_epochs pretrain_CDk seed

Ex.: python run_stacked_rbms_nnet.py 0.01 0 [200,100] 0.01 10 1 1234
```

The script `run_stacked_autoencoders_nnet.py` requires the following hyper-parameters :

```
Usage: python run_stacked_autoencoders_nnet.py lr dc sizes pretrain_lr pretrain_n_epochs
       pretrain_noise_prob seed

Ex.: python run_stacked_autoencoders_nnet.py 0.01 0 [200,100] 0.01 10 0.1 1234
```

The scripts will print the errors on the training and validation sets after every epoch of training. At the end of training, the errors on the training, validation and test sets will also be appended into text files named `results_stacked_rbms_nnet_ocr_letters.txt` and `results_stacked_autoencoders_nnet_ocr_letters.txt` (respectively). Each new execution of the script will append a new line. Pre-training uses the number of pre-training epochs (`pretrain_n_epochs`) specified by the user. Early stopping based on the classification error on the validation set determines the number of training iterations for fine-tuning (with a look ahead of 5).

Once your implementation is complete, you can generate results on this *OCR letters* data set to assess the performance of your implementation. Specifically, try to :

- report the classification error rates on the training and validation sets **for at least 15 different choices of hyper-parameter configurations** (don't report experiments only with the RBM or the autoencoder, try both at least once) ;
- illustrate the **progression of the classification error on the training and validation sets**, for a configuration of your choice of the hyper-parameters ;
- also illustrate the **progression of the average negative log-likelihood on the training and validation sets**, for a configuration of your choice of the hyper-parameters ;
- report the classification error rate on the test set **only for the hyper-parameter configuration having the best performance on the validation set** ;
- specify a **95% confidence interval** of the test set classification error.

Good luck !