Project Deep learning II:
Implement RBM, DBN and DNN from scratch

Yunhao CHEN
yunhao.chen@polytechnique.edu

1 Introduction

In this project, RBM (Restricted Boltzmann machine), DBN (Deep belief network) and DNN (Deep neural network) have been implemented from scratch, including their unsupervised pre-training and supervised fine-tuning process. More specifically, DBN can be viewed as a stacking of RBM; DNN is indeed a DBN with one supplementary classification layer.

I've studied the generative performance of RBM and DBN on "Binary AlphaDigits" dataset. Three influencing factors have been considered: the number of characters contained in training set, the number of neurones and the number of hidden layers.

As for the DNN, I've mainly studied the influence of pre-training on the classification error rate on "MNIST" dataset. I compared two DNN: one with randomly initialised parameters; and the other is pre-trained in an unsupervised way. Different neural network settings have been considered, in order to obtain a comprehensive comparison.

2 Code structure

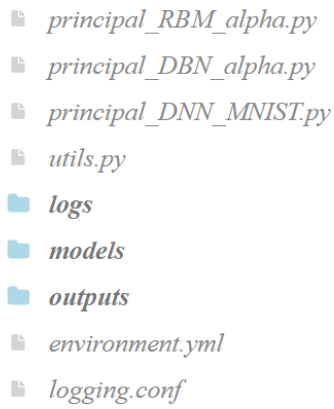

Figure 1: File structures

Figure 1 shows the file hierarchy in this project. "principal_RBM_alpha.py", "principal_DBN_alpha.py" and "principal_DNN_alpha.py" are main scripts. They respectively contains the generative process of RBM and DBN, and the classification process of DNN. The script "utils.py" provides some useful functions. The folder "logs" saves the logging files; "models" contains the pre-trained DNN models; the images generated by RBM and DBN are saved in the folder "outputs". The remaining "environement.yml" and "logging.conf" are configuration files, which provides respectively the Python package dependencies and the logging configurations.

Here are some instructions about how to run the projects. To generate images in "Binary AlphaDigits" dataset using RBM:

```
python principal_RBM_alpha.py --n_characters 12 \
--n_neurons 200 300
```

By running the above command, you will train separately 2 RBMs using 12 characters in "Binary AlphaDigits" dataset. The number of neurons in hidden layer are specified by the argument "n_neurons". In this case, we will train 2 RBMs with respectively 200 and 300 hidden neurons, to generate 3 images (this number is fixed).

As for DBN:

```
python principal_DBN_alpha.py --n_characters 1 \
--n_neurons 300 --n_layers 2 3
```

The above code trains 2 DBNs using 12 characters in "Binary AlphaDigits" dataset. They both have 300 neurons in each hidden layer. The only difference between them is the number of hidden layers (one is 2 and the other is 3). Then each DBN is used to generate 3 images.

For DNN:

```
python principal_DNN_MNIST.py --n_samples 30000 \
--epochs 2 --neurons_per_layer 200 --n_layers 2
```

DNN is trained on "MNIST" dataset. We need to specify the number of training samples, through the argument "n_samples". By default, the DNN would be pre-trained in an unsupervised way (by layer-wise greedy algorithm). You can diable the pre-training, i.e. randomly initialise the parameters, by passing "–no-pretraining". You can also specify the other hyperparameters of DNN by setting the arguments "lr", "batch_size".

Note that you can get the help of how use these scripts by the command "-h". For example

```
python principal_DNN_MNIST.py -h
```

3 Studies on Binary AlphaDigit

In this section, we will discover the influence of several hyperparameters (the number of neurons and layers, the number of training characters) on the generative performance of RBM and DBN.

3.1 RBM

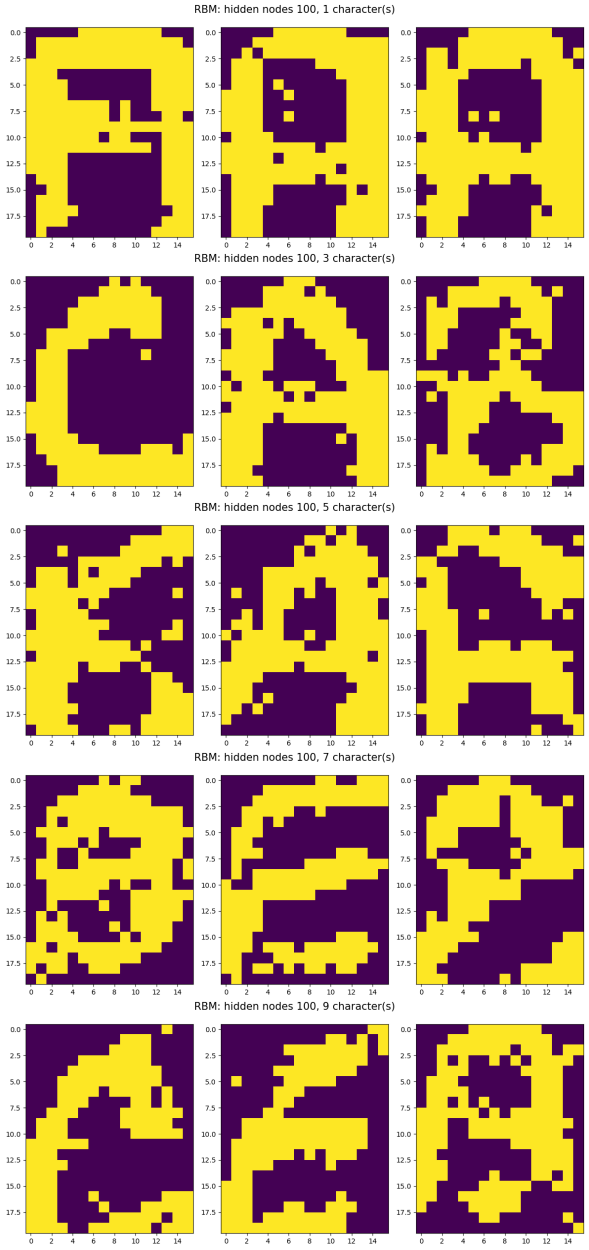

Figure 2: RBM: 100 neurons, with different training characters (from top to bottom: 1, 3, 5, 7, 9)

Figure 2 shows the generated images for different number of training characters when the number of hidden neurons is fixed as 100 in RBM. We found that when we use only one character to train RBM, the generated images could be easily recognized (here it's "A"). If we use more training characters (3 for example), the diversity of generated images increases (we can recognize "C", "A" and "B" respectively). But if the training characters continue to increase (up to 5, 7 and 9), RBM fails to generate recognizable images. The same trend could be found in Fig. 3. When the neurons increase to 400, the express ability of RBM becomes stronger, i.e. it can generate recognizable images with more training characters (up to 7). Consequently, RBM has its limit express ability (represented by the number of training characters). Moreover, it seems that this limit could be loosen when the number of neurons increases.

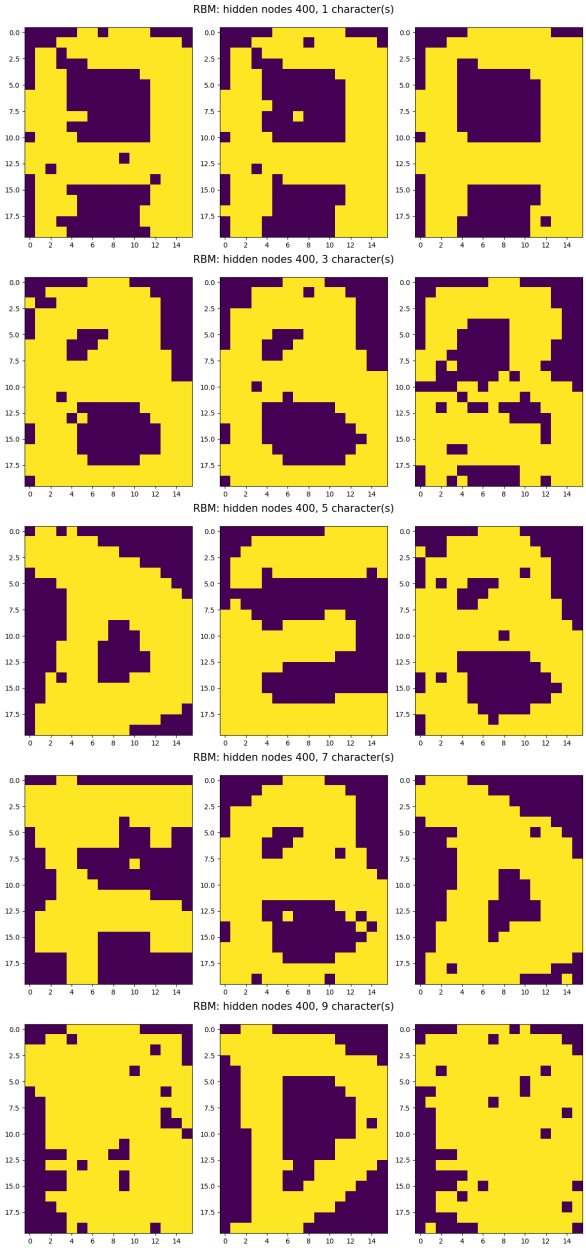

Figure 3: RBM: 400 neurons, with different training characters (from top to bottom: 1, 3, 5, 7, 9)

Figure 4 shows the generated images for different number of hidden neurons when the training character is fixed as "A" in RBM. In this simple generative task, the increase of neurons does not gain a lot on the performance.

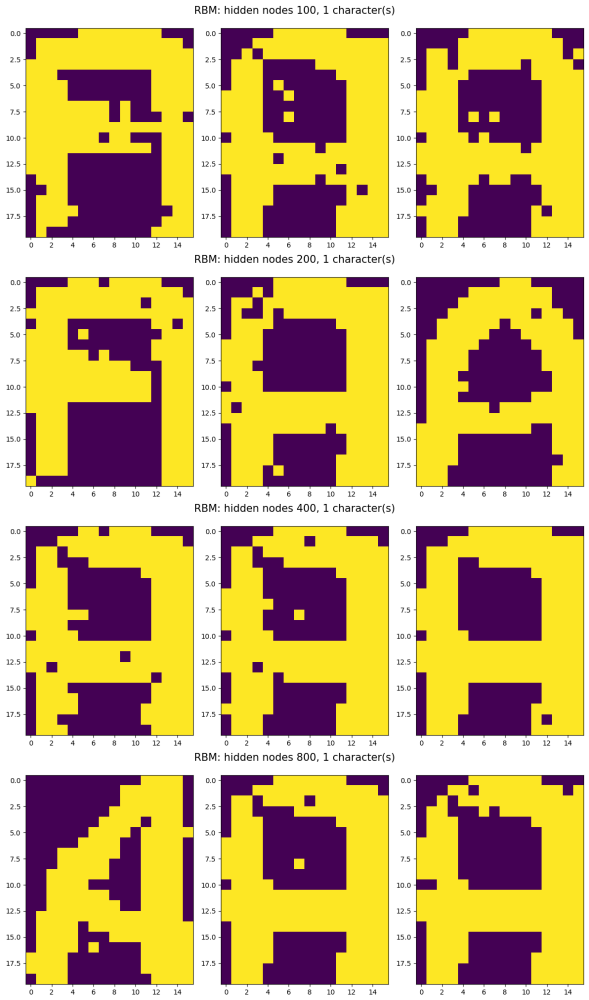

Figure 4: RBM: 1 training character, with different hidden neurons (from top to bottom: 100, 200, 400, 800)

3.2 DBN

We can compare Fig. 3 and Fig. 5. The latter illustrates the DBN with 2 hidden layers (200 neurons in each layer) trained with different number of characters. **The images generated by DBN is much better than RBM**, even if using 9 training characters make decrease its performance. We may conclude that the express ability of DBN is higher than RBM. It is similar with the relationship between perceptron and multi-layer perceptron (MLP).

Then we study the influence of number of layers and number of neurons, as shown respectively in Fig. 6 and Fig. 7. If we fix the number of neurons in each layer (set as 200), it seems that **increasing number of layers do not has a significant effect**. In the other hand, if we fix the the number of layers (set as 2), **increasing the number of neurons from 100 to 800 gains a lot on generative performance**.

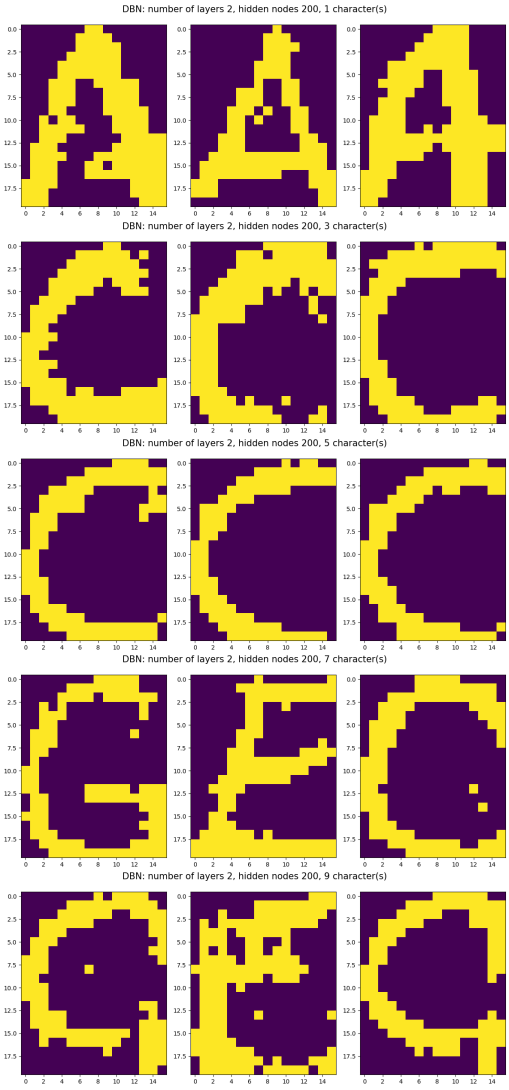

Figure 5: DBN: 2 layers of 200 neurons, with different training characters (from top to bottom: 1, 3, 5, 7, 9)

4 Studies on MNIST

In this section, we compare the classification error rate between the pre-training DNN and randomly initialised DNN.

Globally speaking, **the pre-trained DNN performs the classification on "MNIST" much better than randomly initialised DNN**. Then we study the impact of several hyperparameters. Figure 8 shows that, the error rate decreases when the number of neurons per layer augments. However, there is an inverse trend for the number of hidden layers, as shown in Fig. 9. Another study is carried on the number of training samples (Fig. 10). We can observe that the overfitting problem is alleviated when more training samples are used.

In this experimental setting, **the best configuration for classification is: 2 hidden layers, each with 700 neurons, using all training samples**.

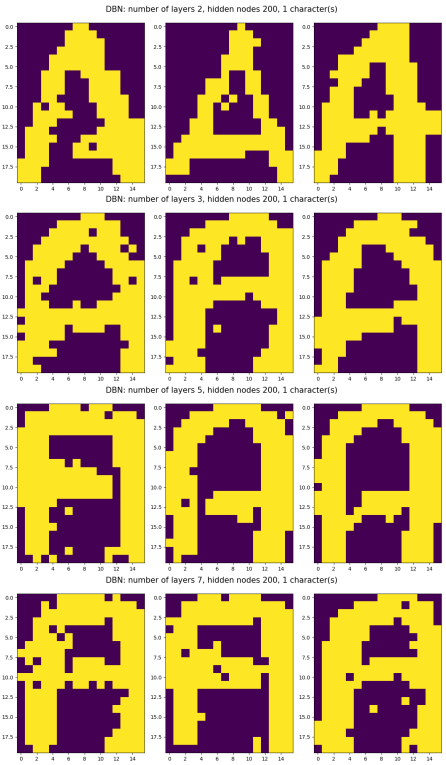

Figure 6: DBN: different number of layers with 200 neurons, using 1 “A” as training character (number of layers from top to bottom: 2, 3, 5, 7)

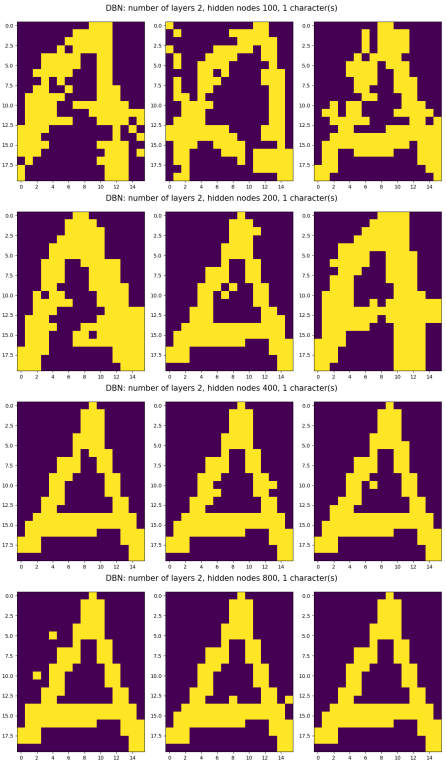

Figure 7: DBN: 2 layers with different number of neurons, using 1 “A” as training character (number of neurons from top to bottom: 100, 200, 400, 800)

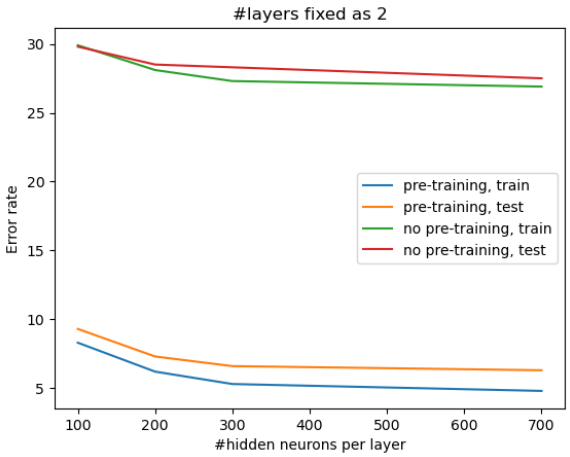

Figure 8: Influence of number of neurons per layer on error rate

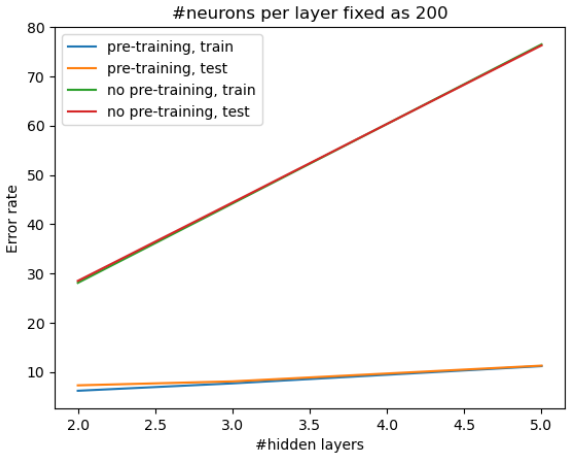

Figure 9: Influence of number of hidden layers on error rate

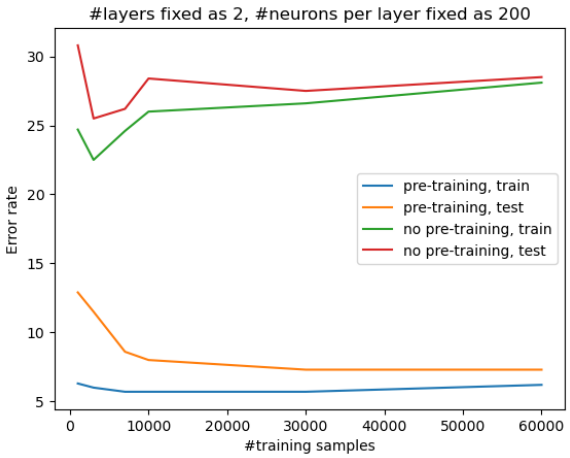

Figure 10: Influence of number of training samples on error rate