M2 Data Science : Deep Learning II

MNIST data classification using Deep Belief Network

Elaborated by :

Seyni DIOP

Khmayes ABOUDA

Alhousseynou BALL

Academic Year 2019/2020

Contents

Intoduction

It is known that the main concern in a learning machine is either to classify or to estimate something. And this happens in most cases by a problem of minimizing often very difficult loss functions (non-convex functions, etc). The appearance of deep neural networks has greatly advanced optimization in this sense and allows to have very good solutions. However, random initialization can sometimes create problems and the pre-training of network weight can be a real advantage. But also explaining how to reproduce the results using the codes provides extra.

Hinton et al. [2] proposed the Deep Belief Network (DBN) which is just a stack of Restricted Boltzmann machines (RBMs) and it is described in their paper that after training the RBM the network can function as a generative network (autocoding among others). In this paper we will try to explain what this consists of, to show if there is improvement or not with unsupervised training(the greedy layer wise algorithm) of the DBNs in MNIST dataset, the performance of DBNs and explain how to retrieve results and graphics in this paper.

1 Restricted Boltzman Machine

A restricted Boltzmann machine (RBM) is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs. Invented by Hilton et al. [3] a RBM machine is an excellent and powerful architecture for dimensionality reduction, classification and collaborative filtering, etc [1]

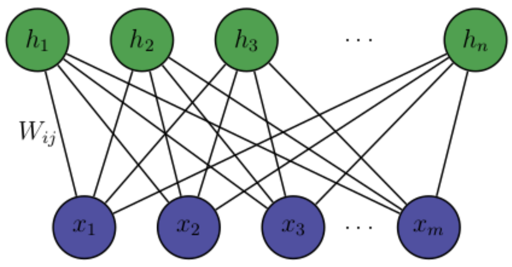


Figure 1: Restricted Boltzman Machine - Source

[1] https://pathmind.com/wiki/restricted-boltzmann-machine

Model training

Given a dataset $\{x_i\}_{i=1}^N$ where x_i is a m dimensional vector. Let's definine training of Suppose there is a Restricted Boltzman Machine with m visible random variables and n latent random variables.

- visible random vector $v \in \{0,1\}^m$
- latent random vector $h \in \{0,1\}^n$
- weight matrix $w \in \mathbb{R}^{m \times n}$
- visible bias $a \in \mathbb{R}^m$
- latent bias $b \in \mathbb{R}^n$

Thus, the energy of the RBM given v and h is

$$E(v,h) = -\left(a^\top v + bh + v^\top Wh\right)$$

The probability distribution of latent variables given visible variables is

$$p(h=1|v) = \sigma(W^\top v + b)$$

where σ is the sigmoid function. And the probability distribution of visible variables will be given latent variables by

$$p(v=1|h) = \sigma(Wh + a)$$

To estimate paramters we may use the convergence divergence algorithm described below

Convergence Divergence(CD) algorithm

1. **Data:** N data samples in $L = \{x^{(1)}, ..., x^{(N)}\}$
2. **Input:** n: number of latent variables, α: learning rate
3. **Output:** learned weights and biais W, b and a
4. init the weight and bias of the RBM
5. foreach $x^{(k)}$

```
    $v^0 \leftarrow x^{(k)}$
    $h^0 \sim p\left(h|x^0\right)$
    $v^1 \sim p\left(v|h^0\right)$

    update the gradient amount
    $\Delta W = \left(v^0\right)^\top \left[p\left(h=1|x^0\right)\right] - \left(x^1\right)^\top \left[p\left(h=1|v^1\right)\right]^\top$
    $\Delta a = v^0 - v^1$
    $\Delta b = p\left(h=1|v^0\right) - p\left(h=1|v^1\right)$

    update parameters
    $W = W + \alpha \Delta W$
    $a = a + \alpha \Delta a$
    $b = b + \alpha \Delta b$
```

2 Deep Belief Network

As mentioned earlier, DBN is a concatenation of RBM layers. It was also introduced by Geoffrey Hinton in their article. When a DBN is trained in an unsupervised way (training of the RBMs that make it up), the RBMs learn how to reconstruct the input data by estimating the probabilistic distribution of the data. How to train the RBM? The pre-training or unsupervised training is done layer after layer. We start by training the lowest layer and when the parameters of these layers are trained, we do the the same thing with the next layer with the outputs of the first layer as inputs, and so on: this is the greedy layer wise algorithm as suggested by Bengion and al.[1].

The DBN can also be trained in a supervised way in classification problems among others. problem that we will deal with in this document, using gradient descent algorithms, etc.

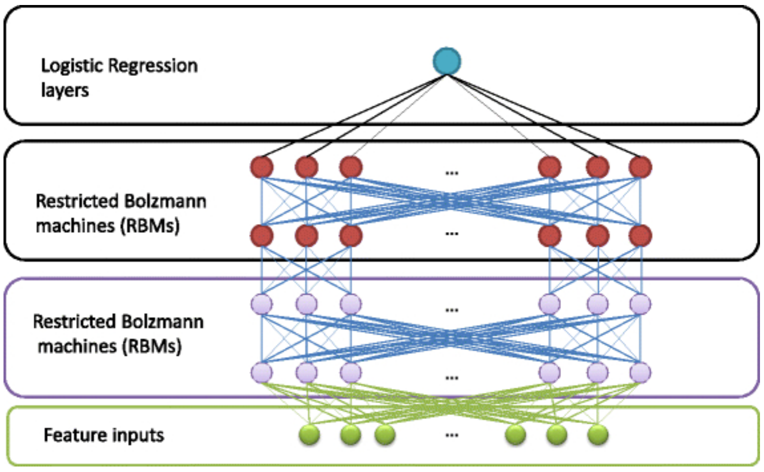

Logistic Regression layers
Restricted Bolzmann machines (RBMs)
Restricted Bolzmann machines (RBMs)
Feature inputs
...

Figure 2: Deep Belief Network - Source

3 How to reproduce results of this paper ?

This document is accompanied by a supplement to be able to reproduce the results as well as to train NDB in a much more general way.

To have a good organization of our main functions, we have separated them into 3 files.

The file `rbm_functions.py` contains all the functions on RBMs namely to initialize an RBM (`init_RBM`), to train an RBM (`train_RBM`), with the CD algorithm described in 1, and to generate images with an already trained RBM (`generer_image_RBM`).

The file `dbn_functions.py` contains the functions to declare DBNs. Since DBNs are concatenations of RBMs, the functions in this file call the functions that are in `rbm_functions.py`. In `dbn_functions.py`, one can find `init_DBN` to initialize DBNs and `train_DBN` to train in unsupervised way a DBN with the greedy layer wise described in 2 as well as `generer_image_DBN` to generate images from DBN.

As for the file `training_functions.py`, it contains a function allowing in a supervised way a DBN (`retropropagation`, see project file for more description).

And finally in the notebook `notebook_project`, you can find all the codes of the graphs and results presented in this document.

4 Results

Please note that the gradient backpropagation algorithm used in this document is Adam. Indeed, during the training it was found that the simple stochastic gradient algorithm stagnated at one point and could not be left.
Throughout the rest of the document, until otherwise stated, we have used this following hyperparameters:

- batch size: 128
- adam parameters : $lr = 0.01$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1e - 8$
- number of epochs: 20
- training data size : 6000 (10% of all data)

4.1 Comparaison about number of layers

First of all, we tried to see the importance of pre-training according to the depth of the network. In the graph below, we represent the classification error = **1-Accuracy** according to the number of layers (of 200 neurons each).

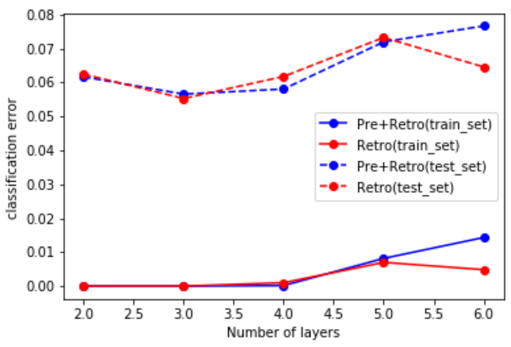


Figure 3: Classification error- by number of layers of 200 units

We can see from this graph that in the views of the hyperparameters indicated above, a 3-layer network seems to give better results and that the deeper the network, the greater the classification error (on the test set). However, the results must be qualified because they depend strongly on the hyperparameters.

L'on notera de plus grande erreur de classification pour le problème lorsque le réseau est à 6 couches, contrairement à ce qu'on attendait.

4.2 Comparison about number of neurons

In this sub-section we have measured the performance of networks with two layers but with different sizes (number of neurons).

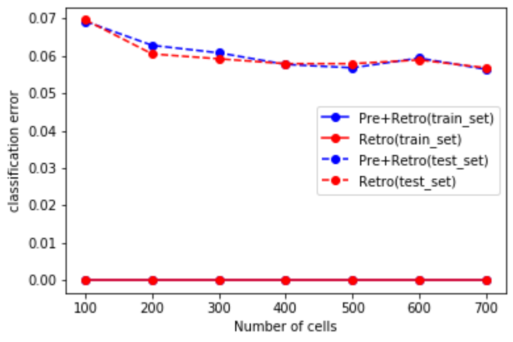


Figure 4: Classification error- by number of of neurons per layer

We can retain mainly from the graph above that globally the more the number of neurons increases the better the results. We can observe also that from 500 neurons the performances are almost only with 700 neurons. And compared to the usefulness of the pre-training, we do not notice significant differences on the final results after 20 epochs (supervised learning for classification).

4.3 Comparison about number of training examples

In this part, we will try to test the performance of a two-layer DBN with 200 neurons each, depending on the number of examples used during training.

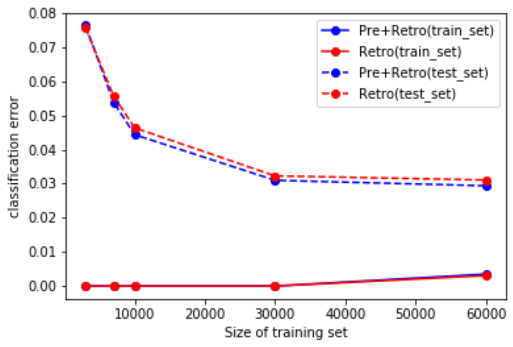
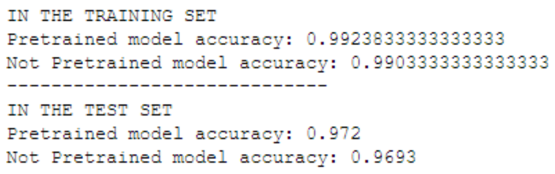


Figure 5: Classification error- by number of training examples

It is clear from the above results that having more examples in learning strengthens our knowledge base. There are better results, on a test basis, with more and more data in the training. Thus better performance can be expected with a neural network that is well supplied with data during training. Compared to the usefulness of pre-training here, one can notice a slight performance increase of the pre-trained networks as the number of data increases.

4.4 A possible best model ?

The performances of a network depend a lot on the hyperparameters used during the training however in our case we can retain from 4.1 and 4.2 that a network with 3 neurons and with 500 neurons at the first layer seems to give better performances, which is in agreement with the results of [2]. The latter will suggest using 2000 neurons at the last layer. This network, trained allows us to have the results below:



```
IN THE TRAINING SET
Pretrained model accuracy: 0.9923833333333333
Not Pretrained model accuracy: 0.9903333333333333
-----------------------------
IN THE TEST SET
Pretrained model accuracy: 0.972
Not Pretrained model accuracy: 0.9693
```

Table 1: Accuracy of our best DNN (784 $\rightarrow$ 500 $\rightarrow$ 500 $\rightarrow$ 2000 $\rightarrow$ 10) - 60000 training examples, Adam parameters : $lr = 0.01$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ - nb epochs=50

5 Conclusion

DBNs seem to give good performance for the classification of MNIST hadwritten digits. In our results we did not find a huge difference between pre-trained and untrained networks. However we see a small advantage. Looking at the figures in the appendix we can see that the pre-trained networks have smaller losses at the beginning of the gradient backpropagation. However, as the optimization algorithm converges, both end up with the same results.

References

[1] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. pages 153–160, December 2006.

[2] Simon Osindero Geoffrey E Hinton and Yee-Whye Teh. A fast learning algorithm for deep belief nets. 2006.

[3] Ruslan Salakhutdinov Geoffrey Hinton, Andriy Mnih. Restricted boltzmann machinesfor collaborative filtering. 2006.

Appendices

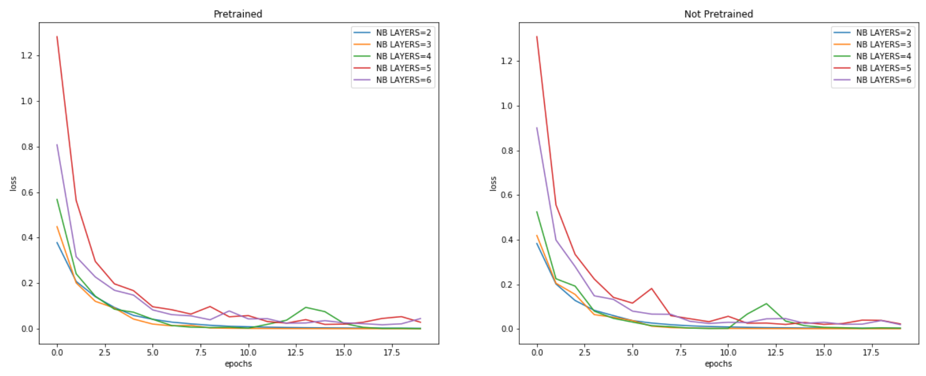


Figure 6: Loss during training - by number of layers of 200 units

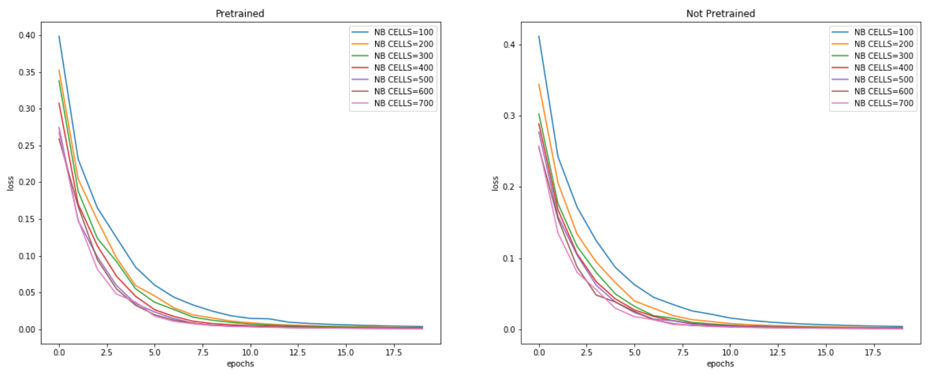


Figure 7: Loss during training - by number of neurons per layer

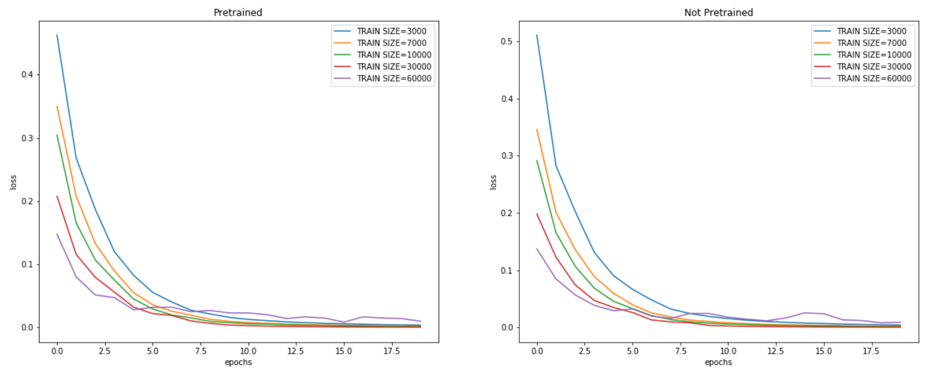


Figure 8: Loss during training - by the number of training examples