Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 4

Restricted Boltzmann Machines and Deep Belief Networks

Alessandro Iucci, Florent Monin and Marie Sarbiewski

April 10, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to understand the functioning of Boltzmann Machines and Deep-Belief Networks (we'll denote them respectively RBM and DBN later);
- to know how to parameter them and how they behave in different conditions;

2 Methods

For this lab, we used Python, and most particularly the libraries:

- `numpy` to implement and easily use matrices and matrices multiplications;
- `matplotlib.pyplot` and `seaborn` to draw curves and make visual result;
- the `github` of KTH to be able to work in group and to use versioning.

We did not use scikit-learn, as we used the skeleton given for this lab.

3 Result and discussions

3.1 RBM for recognising MNIST images

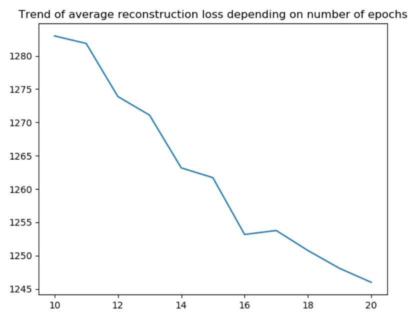

Figure 1: Reconstruction loss trend training network with different number of epochs

For this first task we implemented the correspondent parts of code needed and we different different configurations of hidden units and we also tried to train the RBM for different number of epochs and then comparing the quality of the network. To compare the quality of training we used the reconstruction loss plotted over 11 different number of epochs: from 10 to 20. Even though the reconstruction loss may not be the best indicator, we can see how it decreases with the number of epochs. As we can see at the beginning we have a faster decrease while then it becomes slower.
About the stability, we can see that the reconstruction images after one epoch are already well defined, and going on with the epochs we just have small changes in the probability levels but not in the general shape of the result.

We then investigated how the average reconstruction loss by the number of hidden units decreasing from 500 down to 200 with steps of 25 units per time and we had as a result that a greater number of hidden units leads to lower reconstruction loss

The outcome of the learning network after one epoch are shown in Figure 3: as we can see, we can already start recognising some shapes of some numbers overlapped, even though we can't really distinguish any of them

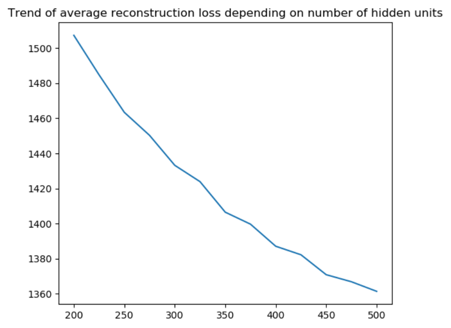

Figure 2: Reconstruction loss trend training network with different number of hidden units

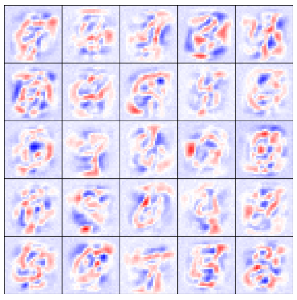

Figure 3: Reconstruction after one epoch

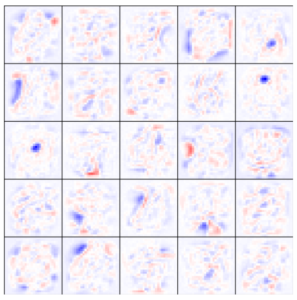

Figure 4: Reconstruction after twenty epoch

3.2 Towards deep networks - greedy layer-wise pretraining

Stack of two RBM According to the proposed architecture, we created a DBN based on a stack of two RBM with 500 hidden nodes, and then we trained it greedily with CD algorithm.

During the training, we reported the loss at each iteration, to see the profile of the loss. As expected, we can see that the loss is decreasing at each iteration. Furthermore, the loss of the second RBM is much lower than the loss of the first one. But we can see that in both cases, the loss is finally converging to a final value, showing a form of convergence in each RBM.

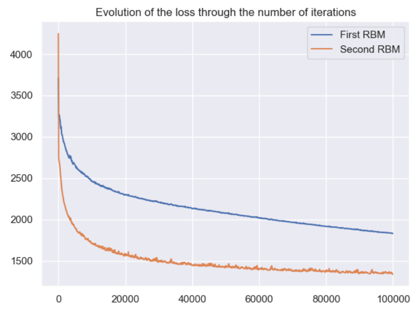

Figure 5: Loss in each RBM

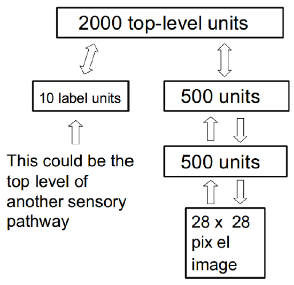

Figure 6: DBN model

Extension of the architecture Once the DBN is pretrained, we extended the architecture by adding a top-level RBM, that allows un to perform recognition and generation of images. We put 2000 nodes on it.

After 500 iterations of Gibbs sampling for an image recognition, we can clearly see the convergence of the model to vector with one 1 (at the index supposed to be the label of the image) and zeros everywhere else. However, the model has to be trained with a lot of iterations : under two epochs, the results aren't very good and the probability distribution computed by our DBN never reaches a stable state. We can see a good convergence in the figure 7. The chosen image is a 1, and the guess on this number was good. However, it wasn't the case for every drawn number in the MNIST database : the best accuracy we could reach was around 85%. The DBN is still quite good with only one or two epochs : the accuracy with so few training can rise up to 70 %.

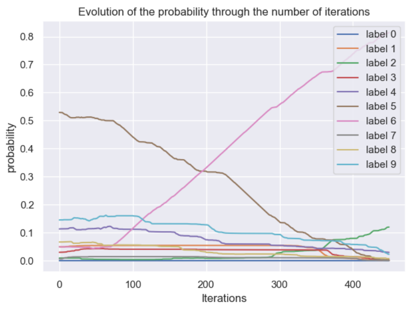

Figure 7: Probability distribution with respect to the number of iterations

Generative mode After the recognition mode, we used the the generative mode, that consists in clamping the label units and then performing at least 200 Gibbs iterations from the top-level RBM down to the visible layer. Then we generated multiple samples for each labels, and stitched them together to make an animation. Generally, we saw that the quality of the image was hanging on the initialisation of the image that we were generating: doing different initialisation when generating images lead us to different results, even with the same pre-trained DBN. Sometimes, the images we got were easily identifiable as numbers for a human observer. Other times, it was harder to guess what number was being generated.

However, the problem that we got during this generative phase was mainly that the generated numbers were right, but not with the right labels. We can then infer that this problem comes from the fact that our accuracy isn't at 100%, and the network learned stable modes, but doesn't really associate them with the corresponding label yet. We can hope to fix this problem in the next section.

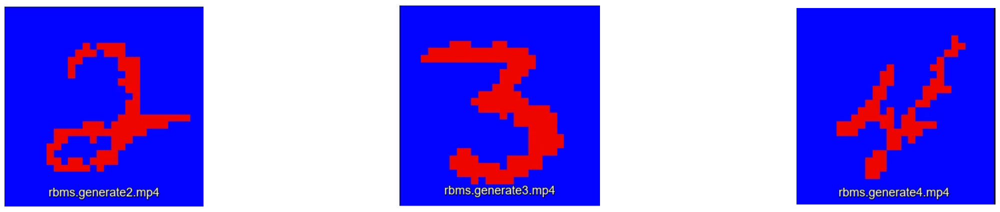

Figure 8: Generation of a few numbers

Here we can see that the generated numbers are very close to some numbers of the MNIST base. This probably comes from the fact that some of the numbers are attractors and are the most-generated ones.

3.3 Supervised fine-tuning of the DBN

We started from the DBN built in the previous section. This way, the greedy pretraining should provide a good starting point for the gradient descent. We used the up-down described in the references. At the beginning, the DBN had a classification accuracy of around 30% (train and test accuracy are roughly the same). We found it was better to underfit the pretrained DBN. When we started with a more pretrained DBN (20 epochs, 80%), the fine-tuning had no effect on the DBN's classification accuracy. With an underfitted DBN (10 epochs), the fine-tuning allowed to go from 30% to 63% classification accuracy, with only 10 epochs of fine-tuning. The training and testing accuracy were always the same, up to 0.1%, which means that this network has a strong generalisation capacity. It should be noted that the whole training (pretraining and training) took around one and a half hour. We can only imagine how many epochs would be needed to get a better accuracy, as our computational resources are very limited. The interest of the fine-tuning was a bit mitigated for us. We can however imagine that, to reach excellent accuracies, the pretraining will be limited in its capacity. Here, the interest of the fine-tuning is much more obvious. Also, the epochs on the fine-tuning were really longer than the epochs of the pretraining, even though this pretraining had to be performed on all layers.

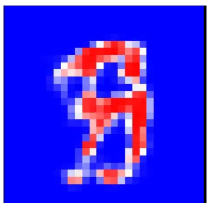
Figure 9: RBMS generation of a 3

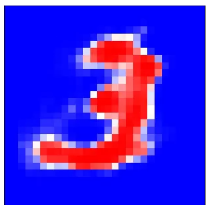
Figure 10: DBN generation of a 3

When looking at the generative results, we had some trouble getting consistent results. It would seem that the images generated depend a lot on initialisation. It seemed to depend heavily on the noise input that was given. We ended up generating random uniform noise between 0 and 1, on a 28x28 grid, that we clamped on the network to make an up-pass, then we did our Gibbs sampling on the top layer. This way, we managed to get pretty decent generations. Both animations (Figures 9 and 10) were really noisy and shaky, and it was sometimes hard to guess the number that was trying to be generated. The network sometimes oscillated between two numbers, especially if they were alike. For instance, when trying to plot a 6, the network really wanted to close the loop, and to go back to a 0, or an 8. To get the results shown here, we had to start from the overfitted network, pretrained for 20 epochs, and trained for 10 more epochs. This means that the classification accuracy didn't increase during the training, but the generation quality slightly increased. The numbers were a bit more shaky after the training, but the general expected form appeared much faster than without training. This might be caused by the model needing more training, as 10 epochs might not be enough to actually converge.

In the last part, we tried to build a network with only two stacked RBMs (`vis` to `pen` and `pen+lbl` to `top`). With this network, even with a large number of epochs (20), the network only reached a low classification accuracy, and the generated images were all random noise, as the activation probabilities were all close to 0.5. It would seem that with this shallower architecture, the network was unable to grasp the underlying structure of the digits. Therefore it couldn't recognise them, or generate them correctly.

4 Final remarks

In this assignment, we learned to build entirely, to set up and to use a Restricted Boltzmann Machine and a Deep-Belief Network. More particularly, we learned how use those models to recognise and generate images with certain labels. We also learned to implement a sleep-wake fine tuning, in order to improve the DBN.

This assignment was different from the previous ones, as we did not only refer to the given subject, but also to some papers to implement all our models. It allowed us to work in more realistic research conditions, and to learn how to select useful/useless information in scientific papers.