Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 4

Restricted Boltzmann Machines and Deep Belief Nets

Filip Husnjak, Ivan Klabucar and Corentin Royer

October 4, 2021

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- apply basic algorithms for unsupervised greedy pretraining of RBM layers,
- to study the proprieties of RBMs on simple image classification problem
- to examine the effect of the number of hidden units and iterations on performance of RBM
- to study trained filters

2 Methods

We used python for all our experiments. We most heavily relied on numpy and its matrix multiplication especially for the implementation of the RBM itself. We also used its np.dot function to compute the dot product of two vectors where needed. To implement the RBM we used the code provided and filled in the blanks where needed. We used either VS Code or PyCharm as IDE. We used github for versioning and a simpler sharing of the tasks.

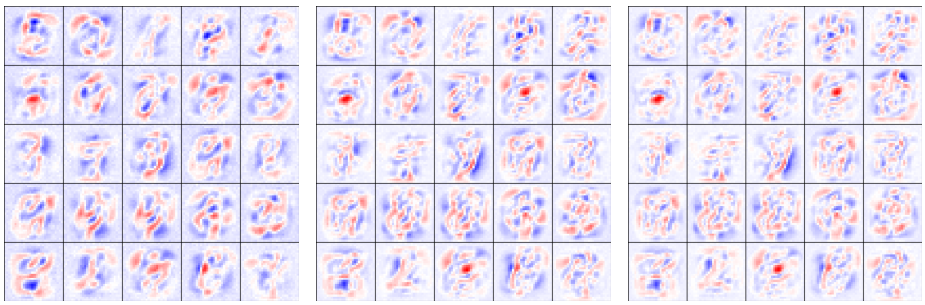
3 Results and discussion

3.1 RBM for recognising MNIST images

In order to build a hand-written digit classifier we first examined the abilities of RBMs to learn suitable features through unsupervised learning. Specifically we

trained an RBM on 60k pictures from the MNIST dataset using the CD_k, $k = 1$ algorithm. For most hyper-parameters we simply used the default ones provided by the given sample code such as learning rate = 0.01, momentum = 0.7, and the initialization of the weights and biases. However, we have examined the effect of the number of hidden units and number of iterations of learning on overall performance. Size of the mini-batch that was used is 20.

On Figure 1 we can see the evolution of certain weights throughout the learning process for architecture with 500 hidden units. For monitoring the convergence of an RBM simply looking at the average reconstruction loss is not enough. That is because reconstruction loss is not the value that CD_k minimizes, nor is it an indicator of the convergence of an RBM. For this reason we also have to examine the weights themselves during training and check that the filters they represent make sense. By reshaping the rows of the weight matrix into 28x28 images we can easily visualize the purpose or rather the filter that each hidden unit provides. When these filters converge to a shape that makes sense we know the RBM is on the right track. In the beginning all of the filters appear to be very grainy (this isn't shown on a figure) because they are randomly initialized. During training, however, they take the shape of red and blue blobs corresponding to regions of the image they respond to or are inhibited by. As training carries on the filters become more abstract and difficult to interpret.

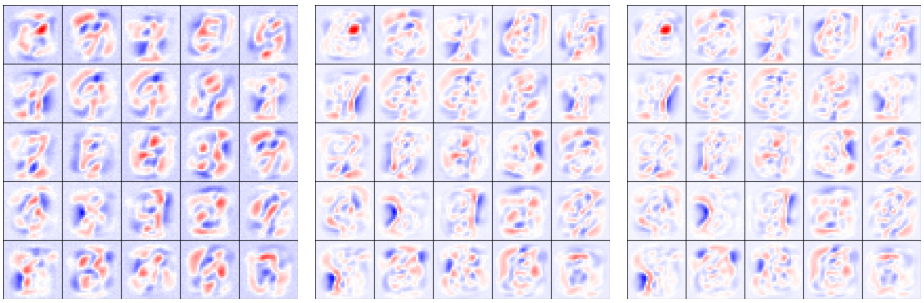


(a) after 5k iter (b) after 25k iter (c) after 40k iter

Figure 1: Visualisation of 25 randomly chosen weights throughout training process for architecture with 500 hidden units.

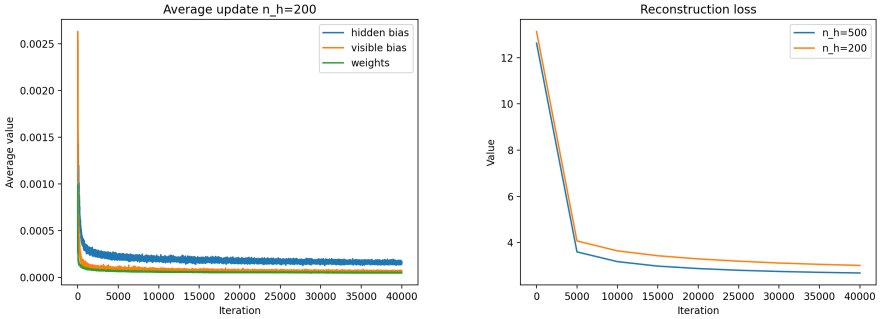
On Figure 2 we can see the same evolution of filters, but for the architecture with 200 hidden units. The difference from the previous case is that the filters are slightly less abstract and easier to interpret. The effect of the filters becoming more complex towards the end of the training is present with this architecture as well.

On Figure 3a we can see that the value of the average absolute update converges relatively quickly to a value close to zero. There is some oscillation from mini-batch to mini-batch but this is to be expected considering the random nature of formation of mini-batches. On Figure 3b we can see the effect the number of hidden units has on reconstruction loss. The architecture with 500 units performs better in this case, although both architectures seem to converge at the same rate.



(a) after 5k iter (b) after 25k iter (c) after 40k iter

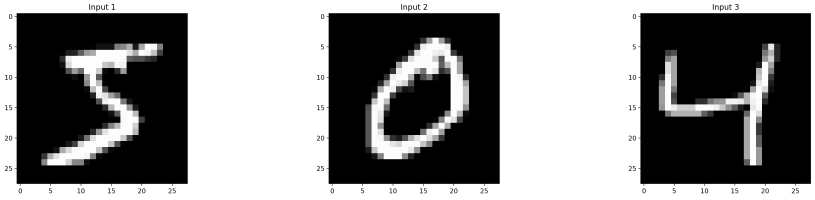
Figure 2: Visualisation of 25 randomly chosen weights throughout training process for architecture with 200 hidden units.



(a) (b)

Figure 3: Convergance of weight updates and effect of hidden nodes on recon loss.

Then we studied the exact reconstruction output of the RBM with 500 hidden units. For this we used 3 images from the test data set as shown on Figure 4.

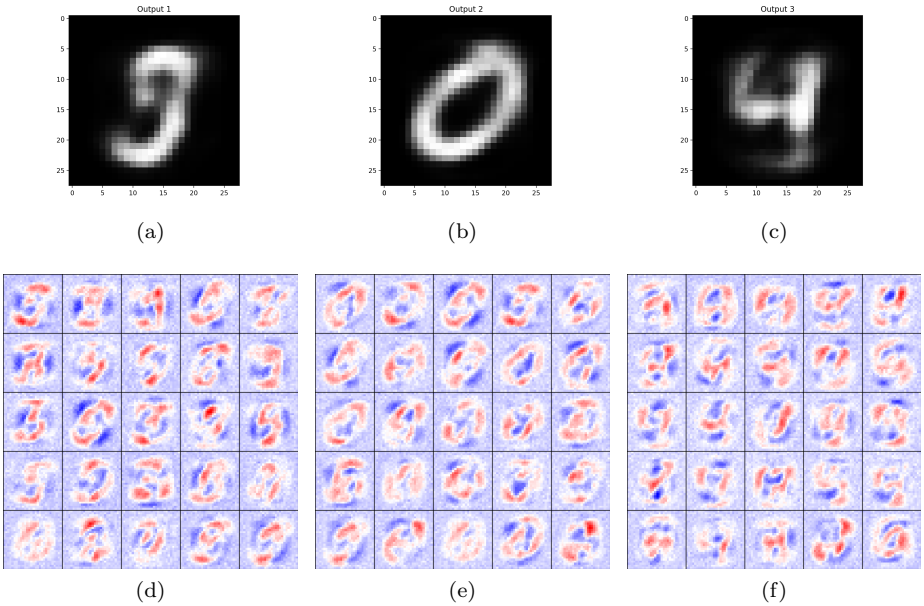


(a) input (b) input (c) input

Figure 4: Three input images for which we examined the reconstruction.

We first examined the output of the network after the 1000th iteration, and computed the reconstruction for the mentioned input images. The results are seen on Figure 5. Below each reconstruction we can also see the visualization

of the 25 most active hidden units during each reconstruction. We can see that for every input image the most active weights closely resemble the desired digit, or at least one of its parts. The reconstructions however are somewhat blurry.

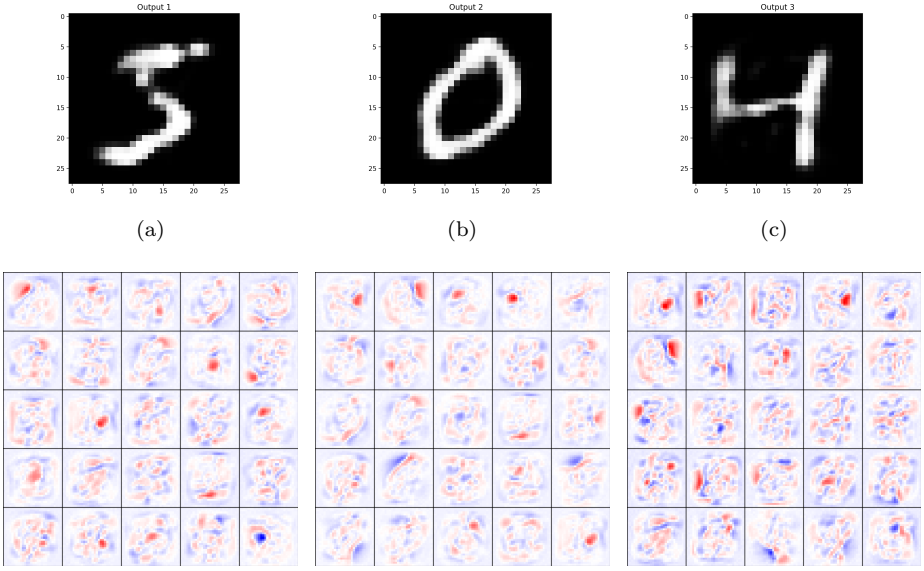


(a) (b) (c)

(d) (e) (f)

Figure 5: Reconstructions after 1000 iters and their respective 25 most probable hidden units

On Figure 6 we see results of the same experiment, but after the entire training process was completed, that is to say after 40000 iterations. The reconstructions this time are much less blurry and more closely resemble the original input images. On the other hand, the most active filters are much more abstract and difficult to interpret than in the previous case. Still, some patterns of the original digits can still be found in the displayed filters.



(a) (b) (c)

(d) (e) (f)

Figure 6: Reconstructions after 40000 iters and their respective 25 most probable hidden units

3.2 Towards deep networks - greedy layer-wise pretraining

Based on the work done in the first part, we implemented a stack of Boltzmann Machines. We trained this stack with a greedy method: we first trained the lower RBM then we freezed it and trained the second one. We did that until we trained the full stack. In our case, we created a stack of two RBMs. On top of that, we added a last layer with the concatenation of the label and the hidden layer of the RBM stack as input. This allowed us to then make prediction and reconstruction on images of the dataset.

During the training, we recorded the loss we obtained for each layer, these losses are compiled in table 1.

Name of the RBM	Loss after 10000 epochs
Top layer	2.39
Penultimate layer	2.76
Bottom layer	3.58

Table 1: Loss for every RBM in the stack

The first task we performed with the network was tha classification of images from the MNIST dataset. For that, we compute the hidden layer one after the other in the RBM stack. When arriving at the last layer, we initialize the label with a uniform distribution of probability. We do a number of iteration of Gibbs sampling on the last layer only. At the end of these iteration we have an

accurate distribution of probability from which we sample the label associated with the input image. The accuracy of the trained network are presented in table 2. The convergence of the last layer when we try to recognize an image is very fast as we can see on figure 7.

Training accuracy	83.90%
Testing accuracy	86.05%

Table 2: Accuracy of the network

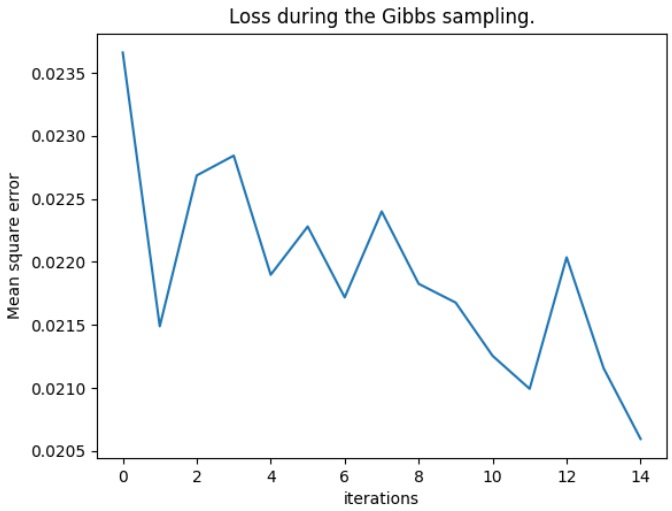


Figure 7: Loss of the network during the Gibbs sampling.

From the accuracies we can conclude that the network has a great generalization capacity, however 80% on the MNIST dataset is far from the state of the art. Moreover, to achieve this result we had to train the network for about 15 minutes.

After that we studied the DBN's perception of images. We fed the network with labels and observed the generated images. First step is fixing the labels in the top RBM layer and generating probabilities for visual and hidden layers of the top RBM network. After that we propagated the probabilities down through the network towards the first visual layer and generated the final image. Results can be seen on the figure below. As one can observe the generated images resemble the expected numbers but the performance is not great and much worse than the recognition performance. We used 200 iterations of Gibbs sampling and the algorithm lasted for about 30 minutes.

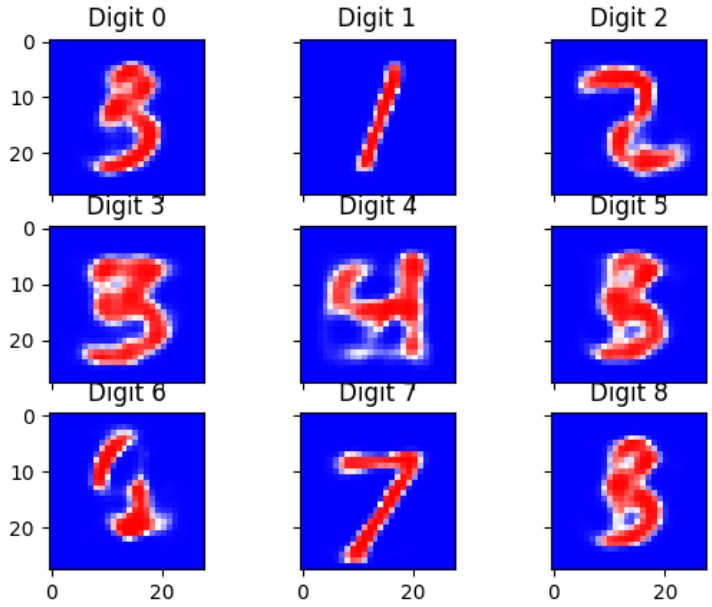


Figure 8: Generated digits using Deep belief network

4 Final remarks

In this assignment we studied the implementation and performance of RBM networks as well as Deep belief networks. We explored the importance of stacking multiple layers of RBM networks creating the deep architecture. We used the networks for pattern recognition as well as to generate patters from labels which can be used for data compression.