Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 4

Restricted Boltzmann Machines and Deep Belief Nets

Emma Palm, Fredrik Kalholm and Javier Lopez Iniesta

February 21, 2023

1 Main objectives and scope of the assignment

This lab was focused on furthering our understanding of restricted Boltzmann machines (RBMs) and deep belief networks (DBNs) with the aims being the following:

- To understand the key ideas behind the learning process in RBMs
- To apply basic algorithms for greedy layer wise unsupervised pre-training of RBM layers to be used in DBNs and fine tuning of the subsequent final layer in the DBN
- Design multi layer neural networks based on RMB architectures for classification problems
- Study the functionality of a DBN including its generative aspects.

2 Methods

This lab was done in Python based in the provided code and data resources. The packages used included, ceil, mean_squared_error and matplotlib.pyplot. The data used was the training and test images and lables from MNIST.

The first part of this lab was focused on training an RBM network to learn the data representations in the MNIST data set. RBMs are networks that have a visible layer v and a hidden layer h. Connecting the nodes in the hidden and visible layer is a weight matrix W with each weight corresponding to a connection between a node in the hidden layer and a node in the visible layer. In addition each node in the hidden layer is associated with a bias. The values of the nodes in these layers can be either 1 or 0 corresponding to an on or off state respectively which is determined based on the probability of the nodes being in the on state p(x = 1).

The network is trained in a forward pass where the p(h = 1) is calculated for each node. This is done according to equation (1).

$$P(h_i = 1|v) = \frac{1}{1 + exp(-bias_{h_i} - v^T W_{:,i})} \tag{1}$$

The values of the hidden layer the become the inputs for a backwards pass where p(v = 1—h) is calculated according to equation 2.

$$P(v_j = 1|h) = \frac{1}{1 + exp(-bias_{v_j} - W_{j,:}h)} \tag{2}$$

The weights are then updated to minimise the reconstruction error according to equation 3. This is done in an iterative process.

$$\Delta w_{j,i} = \frac{1}{N}\sum_{n}^{N}(v_j^n h_i^n - \hat{v}_j^n \hat{h}_i^n) \tag{3}$$

The second part of the lab a DBN was constructed to perform image recognition. DBNs consist of stacked RBN networks with layer trained with supervised learning on top of the RBN layers. In this case the hidden layer of each RBN network is used as the input to train the next layer. Then the labels are clamped to the last hidden layer before supervised learning is done to train the weights from the final hidden layer to the output layer.

3 Results and discussion

3.1 RBM for recognising MNIST images

In this section a Restricted Boltzmann Machine with binary stochastic units has been developed before moving on to the stacking of RBMs into a Deep Belief Network.

Besides, it was trained with a contrastive divergence algorithm CD_1 (weights connecting visible and hidden unit layers) to learn data representations of the images in an unsupervised manner. The visible layer it is composed of 784 units (input images of 28 x 28 pixels), whereas the hidden layer is composed of 500 units. In addition, the dataset used (MNIST) has been divided in mini-batches of 20 units, and 20 epochs. Thus, Figure 1 shows the architecture of this RBM.

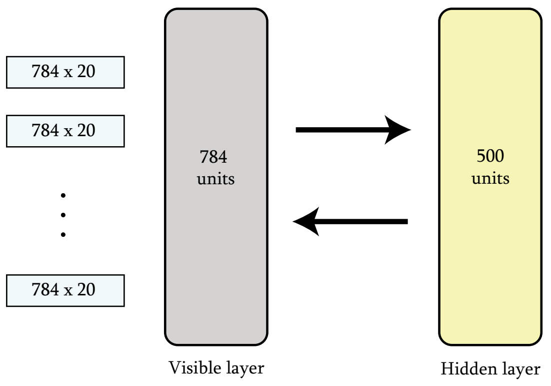


Figure 1: Architecture of the RBM1.

Despite the fact that the RBM models a probability density in an unsupervised manner, we calculated and presented the reconstruction error (which is defined as the mean square error) between each of the 20 epochs as an approximate quantitative measure of the algorithm's performance.

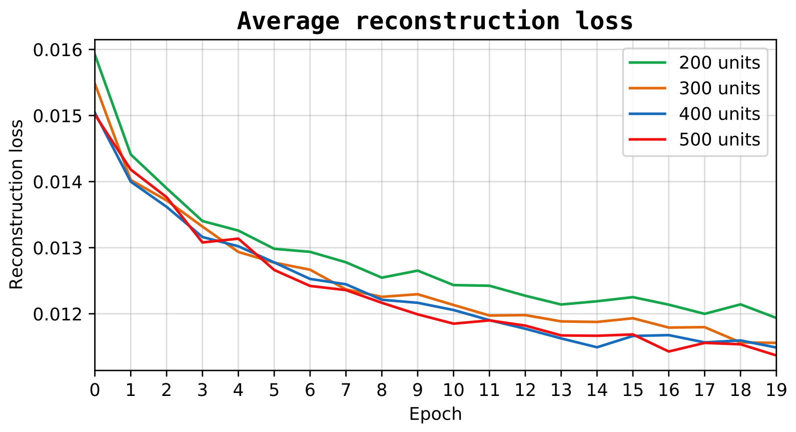
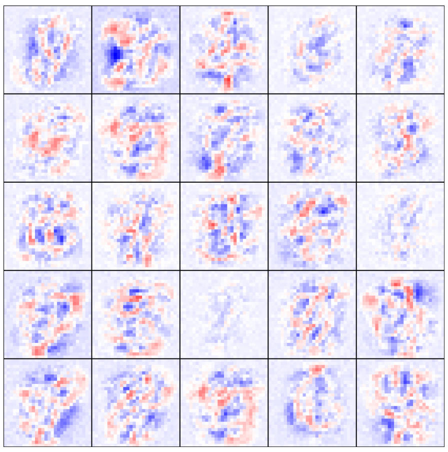


Figure 2: Reconstruction loss over epochs for different number of units in the hidden layer.

Figure 2 clearly shows that the RBM with a higher number of hidden units exhibits lower reconstruction error throughout all training epochs, and it converges to a lower level when evaluated on the training data.

According to [1], it is important to consider multiple factors for monitoring convergence during training, not just reconstruction error. This is because reconstruction error does not necessarily reflect the true objective function that CD1 is improving, and a decrease in reconstruction error does not always indicate that the model is improving [1]. However, while it may not be sufficient on its own, a decrease in reconstruction error can still serve as a necessary condition for convergence: "The reconstruction error on the entire training set should fall rapidly and consistently at the start of learning and then more slowly." [1].

Once the learning process (with 500 units) is complete, we need to analyze the outcomes. Since the training was performed without labels, the evaluation primarily involves assessing the accuracy of the reconstructed images and studying the characteristics of the receptive fields that each unit develops during the learning process. Therefore, we chose to visualize the weights of the visible layer for the hidden units of interest. Each square of Fig. 3 represents the 784 weights of a single hidden neuron.


(a) First iteration.

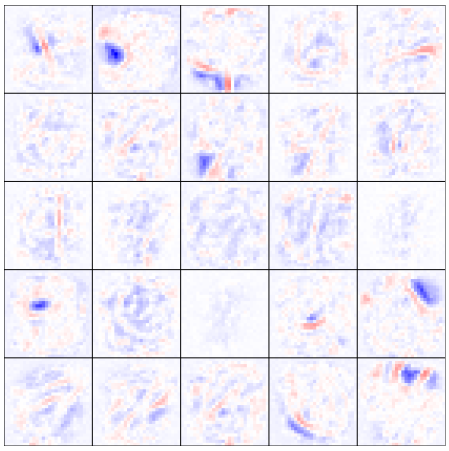

(b) 18th iteration.

Figure 3: Receptive fields.

During the first iteration, the learned filters are quite complex, and one can even recognize digits within them. However, as we progress to the last iteration (18th), the receptive fields become more localized and exhibit stroke-like features. For instance, in the first square, the weight is displayed at the location of the blue and red area, where blue indicates negative weight, and red denotes positive weight. Finally, in Figure 4 the reconstructed image after 20 epochs. Overall, the reconstruction is quite good and it achieves a low reconstruction loss as it can be seen.

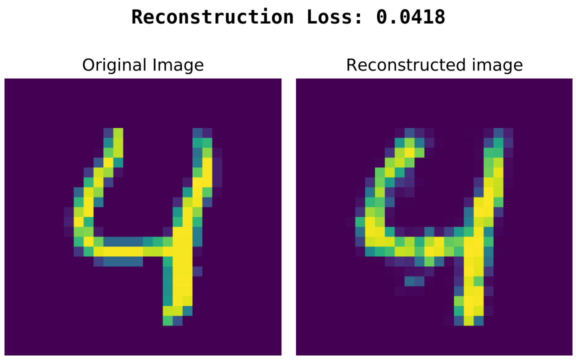


Figure 4: Example of a reconstruction after 20 epochs for the RBM with 500 units.

3.2 Towards deep networks - greedy layer-wise pretraining

This task expands on the architecture set by the previous task to mimic the deep belief network implemented by Hinton et al. in 2006. This architecture uses a flattened version of the input MNIST images as input (784 units) which is subsequently fed to a fully connected hidden layer of 500 units which, in turn, fed to a second hidden layer of 500 units. Here, the 500 units together with the labels of the images (1-hot encoded array corresponding to the digits 0-9) is fed to the top layer consisting of 2000 units.

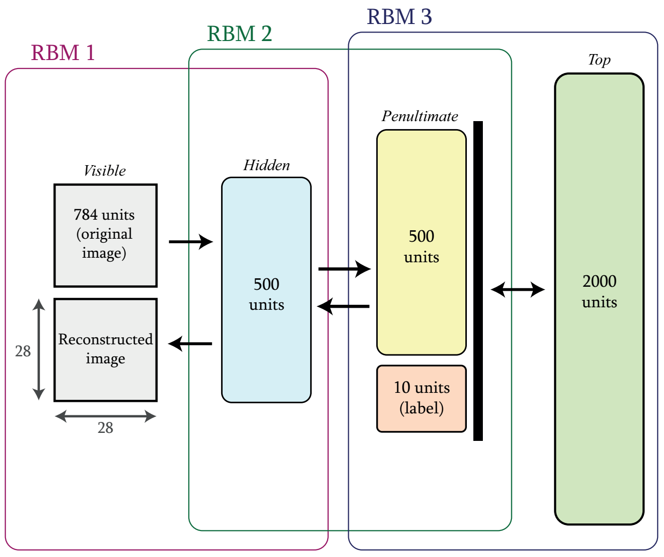


Figure 5: Architecture of the Deep Belief Network (DBN).

Greedy layer-wise pre-training was then performed in each layer, starting from the input to the first hidden layer and then continuing upwards, freezing each hidden layer after each step, which is used as the visible layer for the next training. The reconstruction error for the first and last epoch of each layer is shown in table 1.

Layer	**Loss Iteration 1**	**Loss Iteration 20**
vis - h_1	0.0415	0.0182
$h_1 - h_2$	0.0114	0.0040
h_2 - top	0.0151	0.0042

Table 1: Reconstruction error for the various layers.

After the greedy pre-training was finalized, the final weights in each layer was saved, and the recognition performance (i.e. how many of the digits could be correctly labeled) was evaluated. Here, a binary sample of the input image from the second hidden layer was created, and the corresponding labels was set to 0.1 for each digit. Then, the corresponding representation of the top layer was calculated, which in turn subsequently was used to return the reconstructed version of the second hidden layer + labels. The softmax values of all the reconstructed labels was then evaluated, and the one corresponding to the highest value was the predicted answer. The performance as a function of number of run epochs in the top layer can be seen in figure 6. Then, the corresponding representation of the top layer was calculated, which in turn subsequently was used to return the reconstructed version of the second hidden layer + labels. The softmax values of all the reconstructed labels was then evaluated, and the one corresponding to the highest value was the predicted answer. The performance as a function of number of run epochs in the top layer can be seen in figure 6.

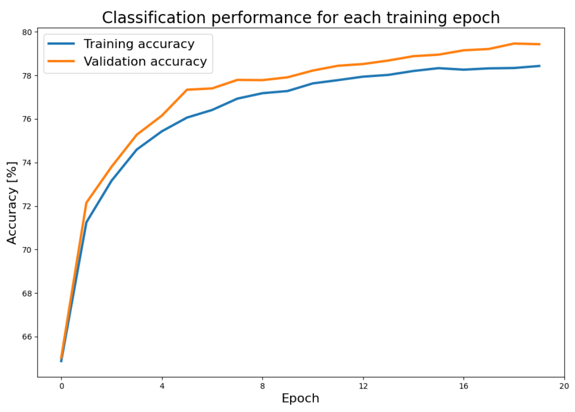


Figure 6: Recognition performance for different epochs.

As can be seen, the performance is very similar for both the training- and the validation set, ending up at approximately 78%, with the validation set even having a slightly higher accuracy. Since we did not use many epochs of calculations in both the pre-training and prediction stages for the sake of computational time, it is not surprising that the performance is similar, as no overfitting trends to the training set have had time to occur. Also, both the training and validation sets were large, which further prevents overfitting.

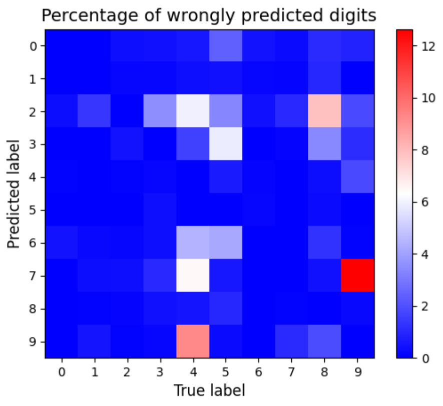


Figure 7: Distribution of missclassified digits.

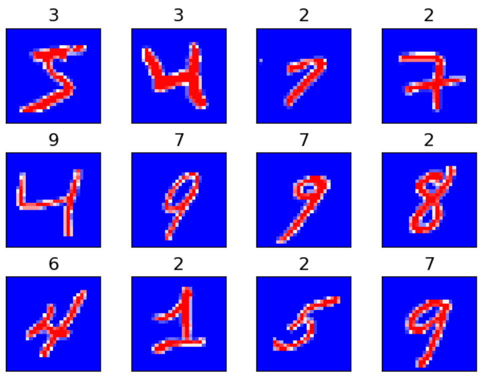


Figure 8: Examples of missclassified digits.

As can be seen in figure 8 and figure 7, the most common mistake of the network was to classify a 9 as a 7, followed by wrongly classifying a 4 as a 9 and a 8 as a 2.

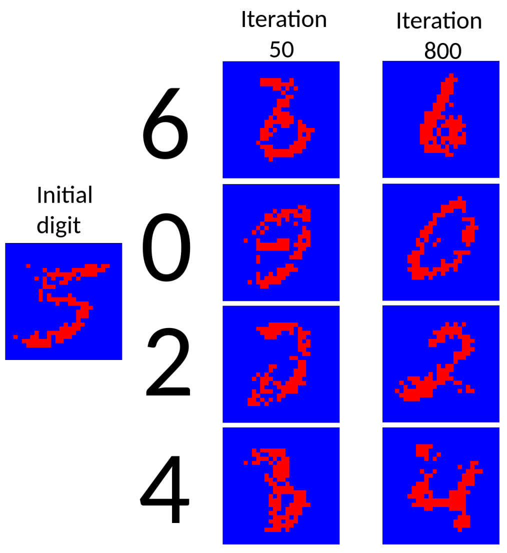


Figure 9: Generated digits for different iterations.

For the next task, the network was utilized to generate new images of digits. This was made by applying the same weights used by the greedy layer-wise training, and then supplying the 1-hot encoded version of the desired digit in the penultimate layer. A random training image was then fed as input, and propagated up to the penultimate layer where is was sampled. This sampling, together with the 1-hot encoded digits was fed to the top layer, where a new sampling of the penultimate layer was made based on the probabilities received back from the top layer. This was sampled again, together with the same 1-hot encoded digits to produce a new result.

Every iteration, the image was also fed down to the bottom layer, and the corresponding visible image was saved. A subset of the results can be seen in Figure 9. In general, the network performed well for generating images. But due to the randomness of the binary sampling process, some individual iterations produced significantly worse results than others.

4 Final remarks

We found this lab to be challenging, as it took a lot of time to understand the general architecture of the python files and what the purpose of every pre-defined function was. As the amount of code was also significantly more extensive than previous tasks, troubleshooting problems was also more time consuming. That being said, we found the lab useful, and it contributed to a deeper understanding of DBN's.

In conclusion, we found that adding more hidden units to the RBM network significantly improved its reconstruction performance. And the reconstructed images from the RBN were also over all good and resembling the input patterns.

For the DBN the accuracy of the model was approximately 78 % for the test set and the model does not seem to be over fitting the data. Last but not least the generative performance of the model was also acceptable and it was able to generate the different digits in most cases.

References

[1] G. Hinton, *A Practical Guide to Training Restricted Boltzmann Machines (Version 1)*, 08 2010, vol. 9.