Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 4

Restricted Boltzmann Machines and Deep Belief Nets

Magnus Tronstad, Teo Jansson Minne and
Quintus Roos

February 25, 2021

1 Main objectives and scope of the assignment

The following report presents the result from the fourth assignment which focuses on Restricted Boltzmann Machines (RBMs) and Deep Belief Nets (DBNs). The main goals for the assignment were to gain a better understanding of the concepts behind RBMs and DBNs, implement and design RBMs and DBNs, and study the functionality of DBNs on recognition and generation tasks.

2 Methods

This assignment is written in Python 3. Computational tasks were done using the Numpy library, and plots were created with Matplotlib library.

3 Results and discussion

3.1 RBM for recognising MNIST images

The implemented RBM was trained for 20 epochs with the Contrastive Divergence (CD) algorithm to learn the 28x28 pixel images of the provided MNIST dataset. The weight matrices for the different numbers of hidden units was initialized with a zero mean normal distribution with $\sigma = 0.01$. Figure 1 shows how the Mean absolute Error (MAE) developed during the course of training the network when using 500 and 200 hidden nodes. The conclusion is that more hidden nodes, in tandem with longer training time, leads to less error.

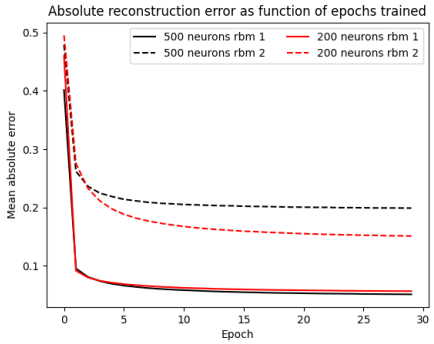

Figure 1: *500 and 200 hidden nodes in the RBM trained with 20 epochs.*

3.1.1 A network of two RBMs

The two implemented RBMs were both composed of 500 hidden nodes and trained greedily layer for layer. The first RBM calculates the reconstruction error based on the differences between its generated reconstruction image and the input image, while the second RBM calculates the error based on the difference between the first RBMs generated reconstruction image and its own generated reconstruction image. The resulting reconstruction error for both RBMs can be seen in figure 1 and it is clearly higher for the second RBM. The smaller network performs better here, likely a consequence of dealing with a simpler representation.

3.2 Training the DBN model

Having monitored the loss curves and receptive fields of a single RBM, we moved on to build up a DBN by stacking RBM:s through greedy training. Since we found that 500 neurons gave better performance, this is what we initially used to construct the DBN:s hidden layers, using the default set learning rate $\eta = 0.01$ and training for 20 epochs (however we also tinkered around somewhat with these parameters, see section Table 1 in Section 3.3).

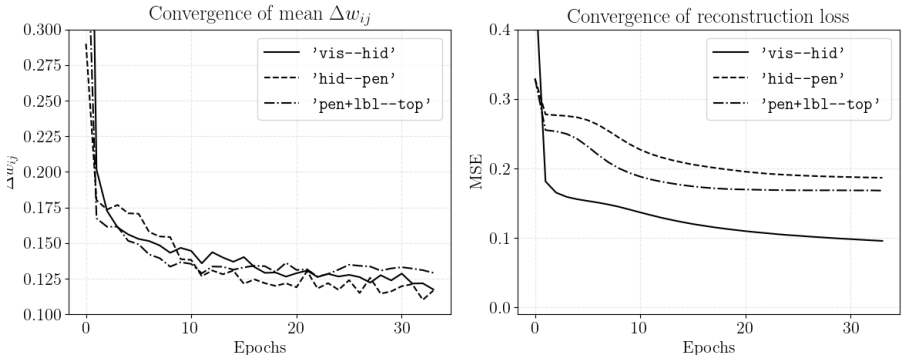

Figure 2: *Plots showing convergence of reconstruction loss (right) as well as the mean absolute value of the weight (+ bias) updates (left) for each epoch of training.*

Figure 2 shows convergence plots of each layer when the network was trained "greedily". Since Contrastive Divergence (CD) does not work to minimize the reconstructive loss, we thought it would also be interesting to record the size of the weight updates throughout training.

From viewing the convergence of the reconstruction loss, it is evident that the layers respond differently to training. The first layer appears to be most apt at reducing the reconstruction loss, and is not done converging after 30 epochs. Clearly, the behavior of the curves do not mirror each other perfectly, which was expected. It is interesting to note that while the reconstruction loss curves differ somewhat qualitatively, the convergence of the weights are more similar in all cases.

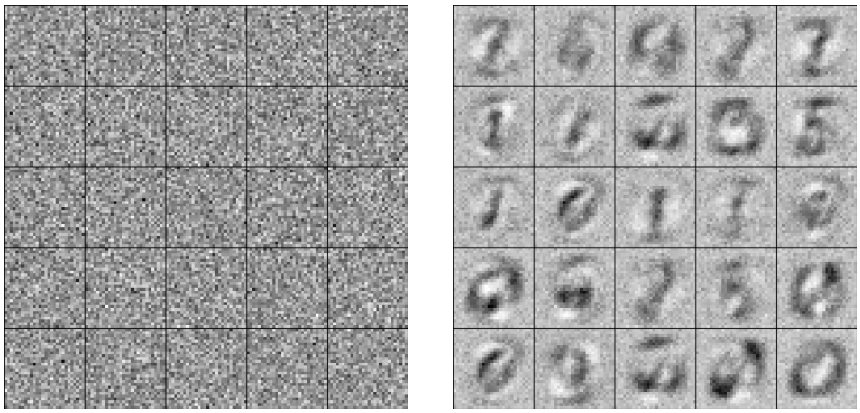

Figure 3: *Representation of receptive fields after 60000 iterations (20 epochs with a batch size of 20) in the first* `'vis--hid'` *RBM layer. Each square plots a row in the weight matrix, reshaped into a matrix.*

3.2.1 Receptive Fields

Figure 3 shows a representation of the "receptive fields" of the neurons in the hidden layer of the RBM, made by plotting 25 rows of the weight matrix, chosen

at random. This type of visualization makes sense since the weights have been trained to literally "weigh" the import of each pixel in the integrated input to a given hidden neuron. The left plot shows the receptive fields in the beginning of training; the right plot after 60000 iterations (20 epochs). Some of the receptive fields are not as clear as others, however, we can definitely recognize the shapes of some digits. In any case there is a stark before and after difference.

3.3 Recognition of digits

Once the DBN networks had been trained they were put to the test. The primary goal for these DBNs were to recognize digits from the handwritten ones present in the MNIST data set, therefore the rate of classification success was the first thing tested. A hold out test set consisting of 10 000 samples was used together with the full training set. When recognizing the digits the network was fed the images to the bottom layer and these were passed up through the network, concatenated with initialized label values of 0.1 at the penultimate layer. Gibbs sampling was then used for 15 iterations and the maximum value of the resulting label probabilities were seen as the classified digit. The results of this can be seen in table 1, showing data from 10 simulations.

Network: N x BS x ET	Rate of success (%) Training set (mean ± sem)	Rate of success (%) Testing set (mean ± sem)
500 x 10 x 7	72.81 ± 0.06	73.22 ± 0.13
500 x 10 x 20	85.46 ± 0.01	86.34 ± 0.08
500 x 20 x 40	87.57 ± 0.03	88.03 ± 0.08
200 x 20 x 20	84.48 ± 0.03	84.91 ± 0.10
200 x 10 x 20	82.98 ± 0.04	83.51 ± 0.10

Table 1: *Digit recognition performance of the DBN. Gibbs sampling using activities. Legend : N = neurons in two bottom RBMs, BS = Batch size, ET = epochs trained.*

Although it is not fair to draw too many conclusions from this test since only one network was trained with each combination of hyperparameters and the stochastic elements of the sampling, initialization and so forth that is present during training needs to be accounted for, we had four key takeaways:

1. The DBN with no fine tuning manages to perform classification reasonably well.
2. The generalization performance to the testing set is not an issue for the network.
3. A batch size of 20 seems to be preferrable to one of 10, note that 40 epochs with a mini-batch size of 20 constitutes the same amount of weight updates as 20 epochs with a mini-batch size of 10. Further testing would be needed to solidify this though.

4. The 200 neuron RBMs performs worse in this regard but not by a wide margin.

It was found that instead of using Gibbs sampling and simply using the probabilities in the penultimate and top layer directly the performance actually increased even though this should remove the mixing capabilities sought after when doing sampling. The results of this, no longer stochastic, can be seen in table 2. Interestingly the best performing network did not remain the same either.
Inspired by the findings in the next section we also checked for classification bias in the networks, interestingly it seemed that the performance on the hardest digits for the network was what separated the better ones from the worst, i.e. the worst performing network had a similar classification ratio of its best digits to that of the best network, however it had some digits with very poor accuracy whereas the better one was less biased. The bias varied between the networks but in general 0's and 1's were always easily classified and 5's and 8's always among the hardest.

Network: N x BS x ET	Rate of success (%) Training set	Rate of success (%) Testing set
500 x 10 x 7	80.02	81.24
500 x 10 x 20	92.06	92.47
500 x 20 x 40	91.83	92.05
200 x 20 x 20	90.03	90.07
200 x 10 x 20	87.97	88.78

Table 2: *Digit recognition performance of the DBN. Gibbs sampling using probabilities. Legend : N = neurons in two bottom RBMs, BS = Batch size, ET = epochs trained.*

3.4 Generation of digits

The significantly harder task of generating digits was also tested.

It was noted that some digits were more often generated well than others, for example 0's were almost always good but 5's, 2's and 8's were a lot harder. A few examples of well generated digits can be seen in figure 4 and some failed ones in figure 5, showcasing some of the most common issues that we found, i.e. making 1's, 7's and 9's indistinguishable (a), being close but not very clear (b) or simply producing a cloud of high/low activity (c,d).

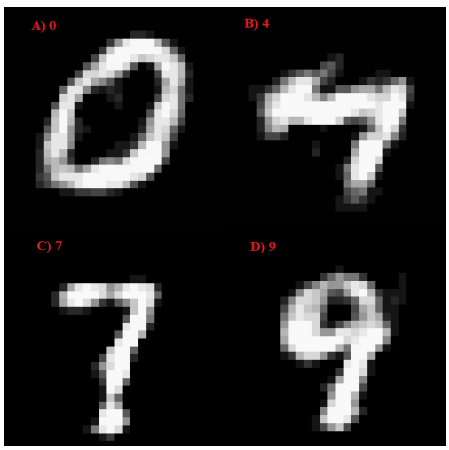

Figure 4: *Decently generated digits.*

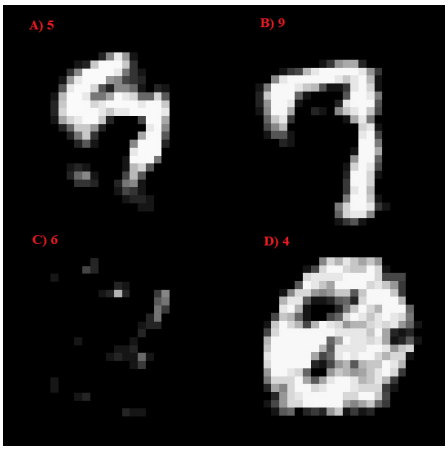

Figure 5: *Poorly generated digits.*

We noted an interesting phenomenon: networks that were very good at generating digits had poor recognition accuracy. For one of the networks that performed well generating digits, the accuracy was only about 55%, and, conversely, one of our best network for recognition was terrible at generation. We got an understanding of this by looking at the bottom layer representations of the receptive fields, comparing one from a network good at classifying to one good at generating digits as is shown in Figure 6.

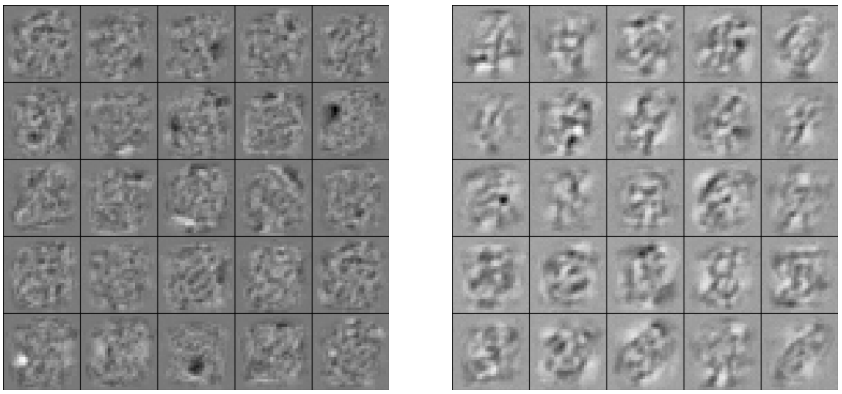

Figure 6: *Left: Bottom layer representations in a high accuracy network (90%+) that is poor at generating digits (trained for 20 epochs). Right: Bottom layer representations in a (90%+) network that is also good at generating digits (trained for 5 epochs).*

3.5 Combining findings to optimize learning

From the results we had gathered we theorized that it was preferable not to train the lower layers for too long, so as to get a representation of the data more akin to the actual digits, but training the topmost 2000 neuron layer for longer improved it's classification accuracy and that this was what caused our tested networks to be specialized in one of the two aspects. With this mind a final network was trained, using 500 neurons in the lower layers, training them for 5

epochs with a batch size of 20. Then training the topmost RBM for 40 epochs with the same batch size of 20. This did indeed create a network more apt at dealing with both tasks, getting an 87 % classification accuracy using sampling and 93 % using probabilities, and still producing a fair amount of legible digits. In figure 7 below all the digits generated from one single run of the network can be seen, interestingly performing well on some of the previously thought hardest digits.

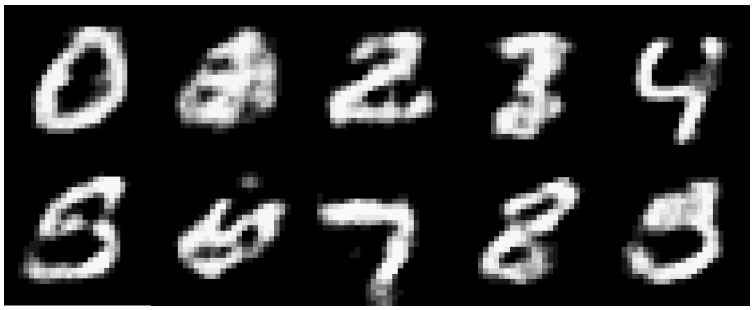

Figure 7: *Generated digits from 0-9 using network discussed in section 3.7.*