Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 4

Restricted Boltzmann Machines and Deep Belief Nets

Patrik Ekman, Eric Hartmanis, Martin Koling and Daniel Wass

September 23, 2020

1 Main objectives and scope of the assignment

This report presents the work and results of assignment 4 in the course DD2437. The assignment concerned Restricted Boltzmann Machines(RBMs) and Deep Belief Nets(DBNs) and our major goals in the assignment were to:

- Gain an understanding of the concepts behind RBMs and DBNs.
- Design and implement RBMs and DBNs.
- Study the functionality of DBNs on recognition and generation tasks.

2 Methods

All code have been implemented using the provided code skeleton in Python3, using Microsoft's developer environment Visual Studio Code as the development platform. Plots have been created using the Matplotlib Python library and computations using numpy.

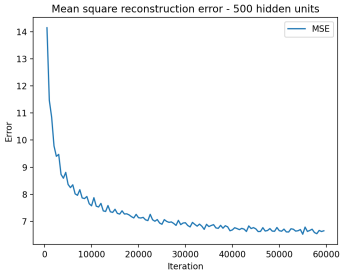
3 Results and discussion

3.1 RBM for recognizing MNIST images

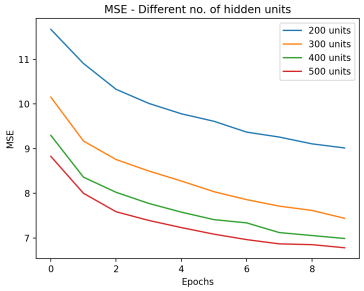
The RBM implemented for this part of the task was trained for 20 epochs with the contrastive divergence algorithm using binary stochastic units to learn data representations of the MNIST data set. The weight matrix was initialized with a zero mean normal distribution with $\sigma = 0.01$ and 500 units was used in the hidden layer. To monitor and measure the convergence and stability in the behaviour of the units a few analyses was made. *Fig. 1a* illustrates the development of the mean square error (MSE) during the training process, evaluated on the entire data set. It is striking how the error descends quickly to begin with, but then slows down the longer the model is trained. This goes well in line with how Geoffrey Hinton describes how the learning process ought to advance and gives a hint of how the model converges (Hinton, 2010).

On top of this, a display of the probability of a subset of the hidden units can be seen in *Fig. 2c*, showing the certainty of the hidden units and how they activate or not. The figure show clear tendencies of randomness, with no obvious vertical or horizontal lines, meaning that most hidden units are being used (no dead units) and that no unusual large or small number of hidden units are being activated. Similar displays was analyzed throughout the training process to ensure that the hidden units behaved as desired. According to Hinton, the randomness involved in the display means that the learning is working properly.

In order to get an overview of how the number of hidden units affected the reconstruction loss, tests measuring the mean square reconstruction error for 500, 400, 300 and 200 units respectively were conducted over 10 epochs with a batch size of 20. As can be seen in *figure 1b*, more hidden units resulted in a better RBM performance.



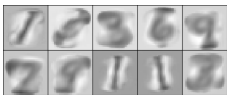
(a) 500 hidden units over 60000 iterations (20 epochs), plotted every 500 iteration.



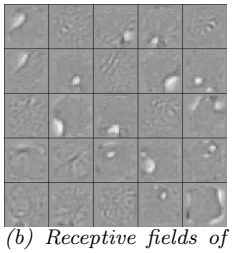
(b) Different no. of units over 10 epochs, plotted after each epoch.

Figure 1: Mean square reconstruction error, with batch size 20.

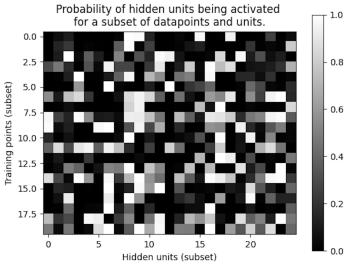
Fig. 2a and *fig. 2b* visualizes weights to the visible layer, i.e the receptive fields of some selected hidden units after 20 epochs of training. *Fig. 2a* showcases the receptive fields for 10 hidden units when using merely 10 units in the hidden layer. In this image, specific numbers can be interpreted easily. This makes sense, since if the number of units are low, each unit will have to learn a large proportion of the data features, i.e. a large proportion of how the numbers in the data images look like. *Fig 2b*, on the other hand, showcases the receptive fields for 25 hidden units when 500 units were used in the hidden layer. Here, the images do not make as much sense. However, this is in line with the previous argument, entailing that the more units that are being used, the less information must be carried by a single unit. Thus, one unit only has to learn some specific small feature of the data, and together they can build a more accurate reconstruction. When training the model with 500 units, the receptive fields in the earlier stages of training was more specific and some of the numbers could even be interpreted. However, the longer the training proceeded, the more general (and more like *fig. 2a*) the visualizations of the receptive fields became.



(a) Receptive fields of all 10 units after training with 10 hidden units.



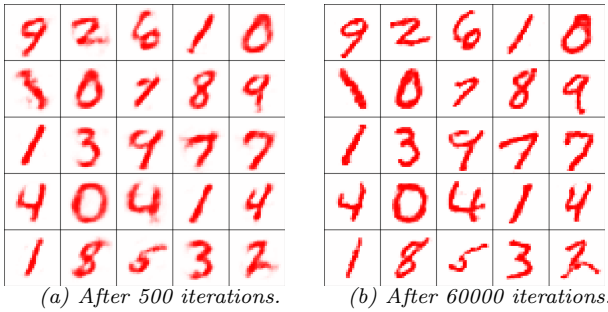
(b) Receptive fields of 25 hidden units randomly selected after training with 500 hidden units.



(c) The activation probabilities for 20 datapoints and the same 25 randomly selected units as in b) after training with 500 hidden units.

Figure 2: Receptive fields and activation probabilities, illustrated by a gray scale where white represents the value 1.

Fig. 3a and *b* illustrates how the implemented RBM managed to reconstruct 25 randomly chosen images during the early and late stages of training, i.e. $p(v_i = 1)$ for $i \in [1, 784]$ for each image. It is evident that the model performs better in the late stage, as the reconstructed images are not as "smeared out", showcasing how the fidelity of the model improved with training.



(a) After 500 iterations. *(b) After 60000 iterations.*

Figure 3: The reconstruction of 25 images.

3.2 Towards deep networks - greedy layer-wise pretraining

3.2.1 A network of two RBMs

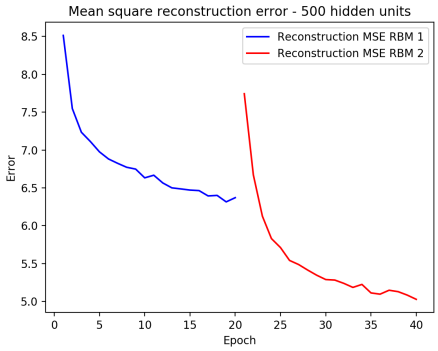
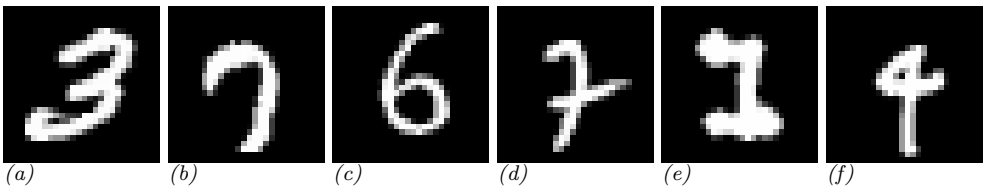


Figure 4: Reconstruction losses for the two RBMs in the stack.

RBM1 is calculating the reconstruction error based on the difference between the input image and the RBM1 generated reconstruction image. RBM2 calculates the reconstruction error based on the difference between the RBM1 generated reconstruction image and the RBM2 generated reconstruction image.

3.2.2 Pretrained DBN for image recognition



(a) *Truth: 3* *Prediction: 8*
(b) *Truth: 7* *Prediction: 9*
(c) *Truth: 6* *Prediction: 0*
(d) *Truth: 7* *Prediction: 3*
(e) *Truth: 1* *Prediction: 2*
(f) *Truth: 4* *Prediction: 9*

Figure 5: Missclassified samples.

The DBN was implemented as in the instructions, with visible units 784, 500, 510 for each layer respectively, and hidden units 500, 500, 2000 for each layer respectively. The CD1 algorithm was used for learning all three RBM layers, i.e. the number of Gibbs sampling iterations k was

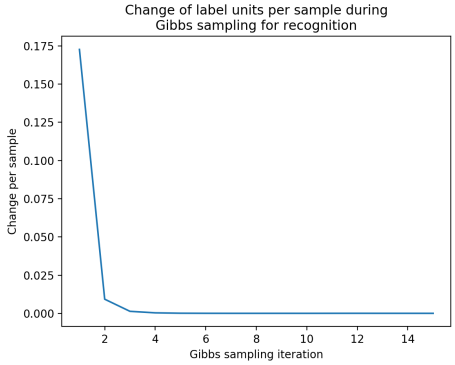


Figure 6: The change in label units for each Gibbs sampling iteration at the top layer during classification.

set to 1. The 500 (for each sample) conditional activation probabilities of the hidden units of the first layer were input for the second layer, and the conditional activation probabilities of the second layer (also shaped $n \times 500$, where n is number of samples) were the input for the top layer.

To let our trained DBN classify the test images, we computed the activation probabilities of the first layer given the image data, and then the activation probabilities of the second layer given the activation probabilities of the first layer, i.e. the bottom-up pass. We then added 10 evenly distributed probabilities for each label, i.e. 0.1 per class. After 15 iterations of Gibbs sampling at the top layer, the model trained over 20 epochs for each layer classified the test images at an 95.61 percent accuracy (average of 5 runs). For the same process with the training images, the accuracy was 96.07 percent (also average of 5 runs.

Figure 5 showcases some miss-classified samples. It is obvious for the human eye that the handwritten digits are not simple to classify, the last '4' could without too much fantasy be a '9' as the model predicted, and the difference between the '3' and an '8' is not very large.

The curve in *Figure 6* shows how much the label units changes for each iteration of Gibbs sampling in the top layer when classifying the test images. It is clear that it converges after just three iterations.

3.2.3 Pretrained DBN for image generation

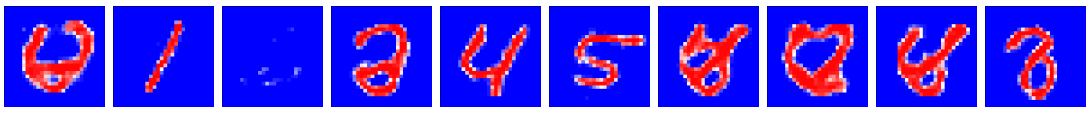


Figure 7: Generated images after 200 iterations of Gibbs sampling in the top layer. The digits 0-9.

For data generation we used the same network as for the classification task, described in the previous section. The data generation was preformed in two alternative ways, both described in (Hinton et al., 2006), however with no clear difference in the resulting visualization. The first was initialized by simply sampling a random input image, with each pixel drawn from an even binary distribution, with the shape 1×784. This image was then taken through the up-pass through the first two layers, and the binary representation of the activation probabilities of the second layer was concatenated with the one-hot-encoding of the label desired to generate. This concatenation was given as input to the top layer, in which 200 iterations of Gibbs sampling were performed with the label representation fixed. This should force the model to activate hidden units corresponding to the desired label. By performing a stochastic binary down-pass of the

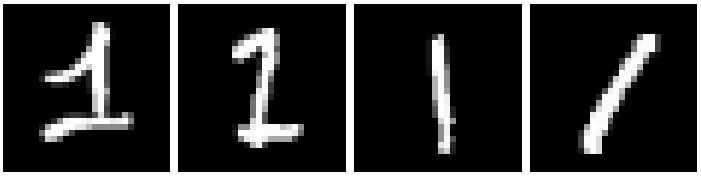


Figure 8: Handwritten ones - a few training samples of the class '1'.

visible units' activation probabilities of the top layer, initialized by a stochastic binary sampling (with the given Bernoulli distribution), an image can be generated at the very bottom of the network by the resulting probabilities of shape 1×784. As the down-pass is stochastic, multiple samples can be drawn from the down-pass without performing more rounds of Gibbs sampling - which saves time complexity.

The second alternative was to skip the up-pass completely and instead initialize the Gibbs sampling at the top layer by a random input of shape 1×500 concatenated with the desired label representation. The rest of the generation was identical.

The quality of the generated images varied much within the classes. For some runs of the generation, they were simple to recognize as the digit they represent for the observer's eye, but sometimes the resulting images were indistinguishable from other digits. The stochastic factor of the generation seemed to play a large role - each image generated from the same Gibbs sampling output differed slightly, but more importantly different complete run, including the Gibbs sampling resulted in completely different looking images. We believe that another important factor for image quality was how much the training images of a class varied. E.g. the handwritten 'ones' of the training set are sometimes diagonal lines, sometimes straight lines, and sometimes also including a bottom line and a top "hook", see a few examples in *Figure 8*. Another data related factor, we believe, is the similarity between some of the images - the difference between a handwritten '9' and a '4' is not always significant, see e.g. *Figure 5f*. Lastly, we observed that the more iterations of Gibbs sampling we performed during the generative process, the higher was the risk of generated images being close to 'empty', i.e. with just a few of the 784 visible units being activated. We believe that this is related to, during the Gibbs sampling, the combinations of visible units together with the forced on (fixed) label pushes the hidden activation probabilities down - thus the probabilities becomes lower and lower for each Gibbs sampling. See, e.g., the third image in *Figure 7*, sampled as the digit '2'. This last factor is of course also tightly coupled to the stochastic factor.

4 Final remarks

This assignment was very interesting, in our opinion. The results were concrete and the literature attached was good.

However, we had much reading, reasoning and experimenting - back and forth - of where to use the stochastic binary samples and where to use the real valued probabilities. Unfortunately, we did not have time to do the optional task (4.3).

References

Hinton, G. (2010). A practical guide to training restricted boltzmann machines.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.