Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 4

Restricted Boltzmann machines and Deep Belief Networks

Christian Ulmer, Zakaria Nassreddine and Simon Langrieger

October 12, 2021

1 Main objectives and scope of the assignment

This lab extends our perspective on Neural Networks (NN) by examining Restricted Boltzmann Machines (RBM) as a pathway into Deep Belief Networks (DBN). For RBMs, we use the contrastive divergence algorithm and inspect the results we can obtain when applying it to the MNIST image dataset. We then build on top of that architecture to implement a DBN and examine the greedy layer-wise pretraining approach and the results it bears both in terms of classification and generation.

2 Methods

This lab was conducted using Python and the framework provided by the assignment authors. We used the `numpy` library for matrix computations and mathematical functions, and `matplotlib` for plotting.

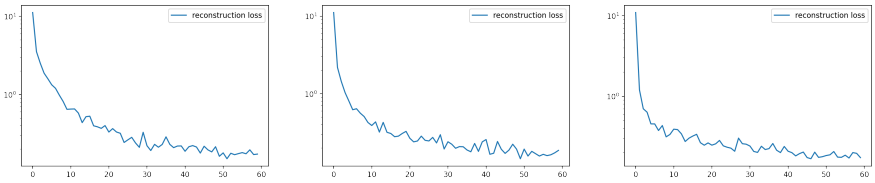
3 Results and discussion

To reach our final goal, which is building and training a functioning Deep Belief Network (DBN) we first have to start with its building blocks which are Restricted Boltzmann Machines (RBM).

3.1 Restricted Boltzmann Machines

Data First however we look at our data which is the very popular dataset MNIST consisting of 60k training and 10k test grey-scale images of handwritten digits together with their corresponding label. Our goal is finally to classify and more interestingly generate new handwritten digits.

Convergence measure The goal of training a RBM is to learn the data distribution of the input data. To do this, images are reconstructed through an information bottleneck similar to an autoencoder. However, instead of reducing dimensionality we sample from a learned probability distribution. Finally this learned internal probability distribution should approximate the data distribution. In autoencoders we use the MSE between input and reconstruction to investigate convergence of the network. The intuitive approach would to use the same measure for RBMs. However, literature suggests that this does not suffice as the only measure. Nevertheless, we will still investigate it. First we looked how the mean reconstruction loss depends on the number of hidden units.

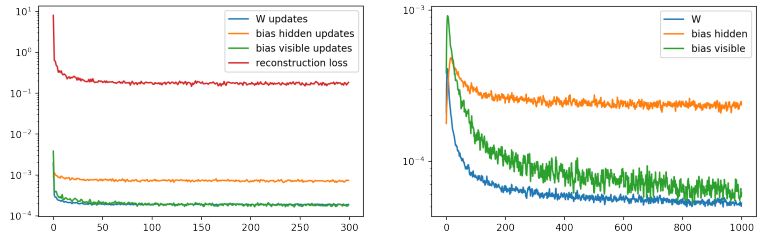


(a) 200 nodes **(b)** 350 nodes **(c)** 500 nodes

Figure 1: Development of reconstruction loss for different number of nodes in the hidden layer

For this we trained 3 different networks for one epoch and plotted the mean reconstruction loss every 10 updates. These plots can be seen in Figure 1. We find that the loss is pretty similar at the end of training - however the loss drops faster the more nodes are in the hidden layer.

We also found why reconstruction error does not suffice as a convergence criteria. However, we experienced this when we generated images with the DBN. We will still point it out here. Even though after a certain point the reconstruction loss stays the same and only jitters a little bit, we found that it still helped a lot to continue training. This improved the generation quality a lot and also helped escaping some error cases. We also looked at the size of the weight updates which however developed just as the reconstruction error did. This can be seen in Figure 2a.



(a) Development of reconstruction error and weight updates **(b)** Closer look at weight updates at the beginning of training

Figure 2: Development of various measures

In Figure 2b which is the same data as in Figure 2a but "zoomed in" in the beginning of the training we see the effect of momentum, which we added to speed up training. In the first couple of updates we really see how the training

first takes up momentum before decreasing the update size again later in the process.

Weight matrix We found the best way to look for convergence was to directly look at reconstructions and the weight matrix itself. If the weights stops changing a lot we know that there is probably no reason to continue training.

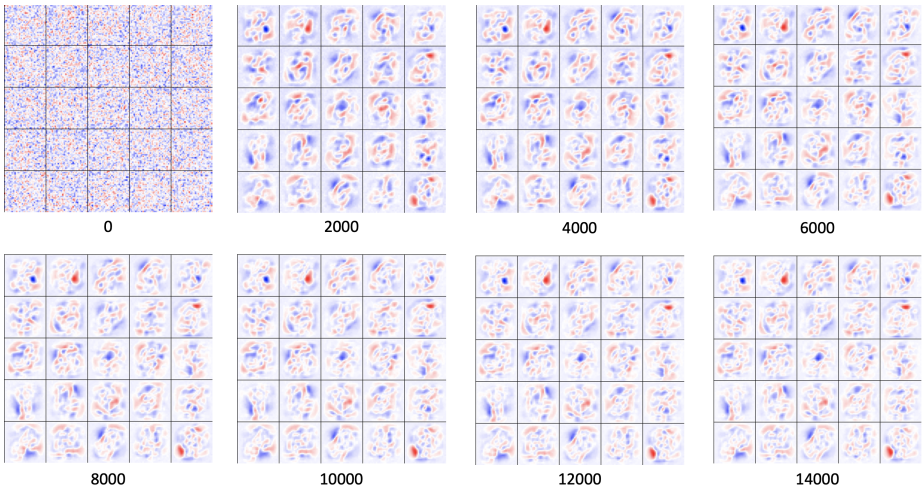
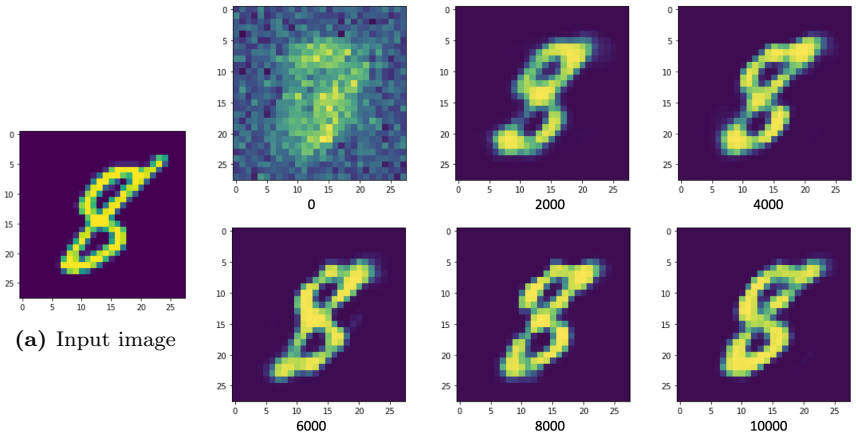


Figure 3: Development of weight matrix of some random notes during training

What we do is, choose random hidden units and look at the connections to the visible nodes reshape them into the same shape as the input data, so 28 by 28 pixel images. These plots can be seen in Figure 3 where we plotted the weights every 2000 updates. We see that after the 6000th update the weights only undergo some minor changes.



(a) Input image

(b) Reconstructed images plotted every 2000 updates

Figure 4: Overview over reconstruction over time

Reconstruction of images Nevertheless, when we look at the fidelity of the reconstructed images (Figure 4), we see somehow different behaviour. Here the images still change significantly and also in the correct "direction" of the original input. The reconstruction after 10000 updates is definitely better than the reconstruction after 6000 updates.

3.2 Towards deep networks - greedy layer-wise pretraining

Two-layer Deep Belief Net We started with a two-layer network (two RBMs in the stack following a 784-500-500 architecture), greedily training one layer after the other. After one update, we report a reconstruction loss of 3.5222 for the top layer, and 8.0133 for the bottom layer. After 5000 iterations, the error drops to 0.5334 for the top layer, and 0.1971 for the bottom layer. We believe this difference of performance is due to the nature of data. While the bottom layer receives very sparse data as inputs, the top one receives the features of the digits which are not as sparse and more "random" which makes it harder to reconstruct.

Three-layer Deep Belief Net In the following section, we extended our architecture to a three-layer network (784-500-500-2000). First we used it to classify input images. For a batch size of 10 over 2 epochs, we get an accuracy of: 90.33%.

Some of the misclassified samples are shown in Figure 5 below, with the following convention: true label → predicted label.

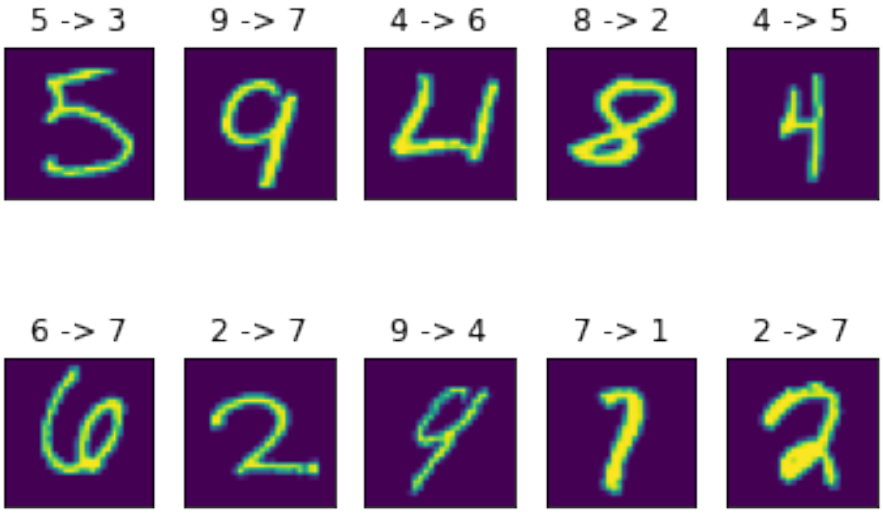


Figure 5: Some wrongly classified samples using the pretrained DBN.

We now proceed to using the DBN in generative mode. We observe that the key factors that play into the quality of the generated images are the settings of the RBM pre-training which is to say the batch size and number of epochs

during training, as well as the number of iterations in Gibbs sampling during the execution of the CD algorithm. While increasing the number of iterations is time consuming, it leads to less noise in the generated images.

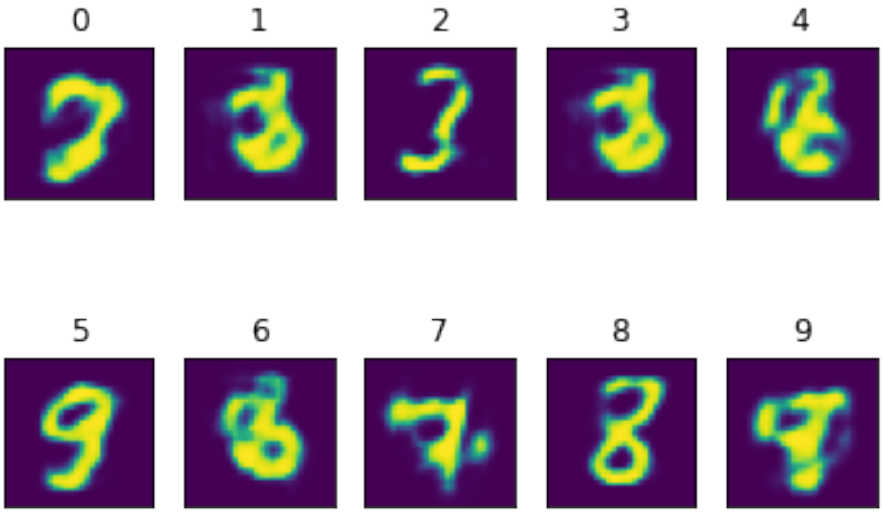


Figure 6: Generated images using the pretrained DBN. Each image is the output corresponding to the input label in the title.

Generated images (cf. Figure 6) are of poor quality and do not resemble the original input patterns. We observe that the network sometimes cares little for the input label, as in the generated output for a given input tends to be closer to another random label. Some generated images illustrate the overlap of different input patterns. Our explanation is that the greedy layer-wise approach only addresses the classification quality by propagating the weights upwards, and does not really work on improving the generation quality.

This can be mitigated with supervised fine-tuning of the DBN and driving the network top-down to improve its reconstruction of previously seen data.

4 Final remarks

This lab was an extensive learning experience that introduced us to deep belief networks, but the massive theoretical material was very hard to cover which we think was a shame because it would have been interesting for us to fully grasp the content were we given more time. The provided supporting resources were very heterogeneous (scientific literature, course slides, a video and a Q&A) and we think that more efforts could have gone to synthesising the material to align with the short period of time (1 week) that we were given. The (voluntary) supervised fine-tuning approach aimed at improving the classification accuracy would have been very gratifying to implement if we had had more time, especially knowing its theoretically anticipated impact on the generation quality.