Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 4

Restricted Boltzmann Machines and Deep Belief Nets

Boyu Li, Tobias Bezner and Nik Joel Dorndorf

February 26, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were to

- first understand how RBMs and DBNs work and how their training is performed
- implement both of them into the code construct
- run various experiments to observe their performance

We tried to implement everything in the most efficient way possible. Anyway, some experiments took a long time. Therefore, we had to do some simplifying at some parts of the assignment.

2 Methods

We implemented everything in Python. We used PyCharm as scripting environment and Github to share our code. In this assignment we implemented all functions concerning the ANN ourselves and did not use libraries. For this reason, we only used standard libraries like numpy and matplotlib.

3 Results and discussion

3.1 RBM for recognising MNIST images

We initialized the weight matrix with small random values (normally distributed: N(0,0.01)) and iterated the training process (CD) for 20 epochs. The size of

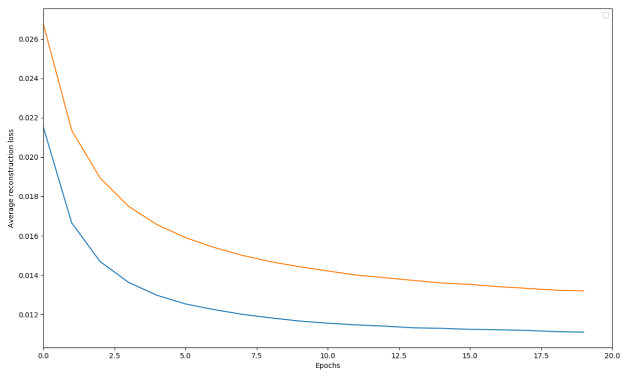

Figure 1: Average reconstruction loss: $n_{hidden(blue)} = 500$, $n_{hidden(orange)} = 200$

minibatches is 20. To monitor convergence or stability, we can stop training when MSE between the original and reconstructed images or ΔW is not larger than a threshold.

From Figure 1, we can see that decreasing the number of hidden units increases the average reconstruction loss.

From Figure 2, we can see that the receptive field is like a mixture of some numbers.

3.2 Towards deep networks - greedy layer-wise pretraining

To limit training time we decided to perform a small analysis of the influence of the batch size on both training time and performance. As one can see in Table 1 increasing the batch size led to a big drop in performance why we decided to stay with a batch size of 20 for the rest of the assignment.

Batch Size	20	40	60
Training Time (sec)	603	510	483
Test Accuracy	80.01%	69.33%	65.63%

Table 1: Greedy Layer-wise Training: Training Time and Test Accuracy over different batch sizes (5 epochs each CD1)

In Figure 3 one can see the reconstruction loss of the different RBMs over 20 epochs. We can see that for all of them the loss goes down strongly in the first 5-8 epochs and then converges slowly. The reconstruction loss increases with the ”highness” of the layer meaning that the input RBM has the lowest one and the top RBM the highest.

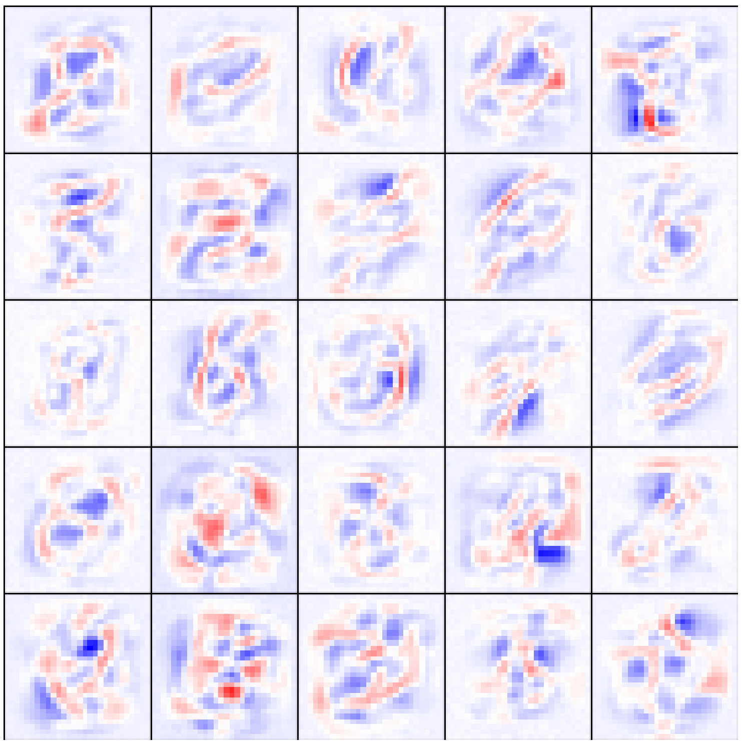

Figure 2: Receptive Fields (15 epochs, $n_{hidden} = 500$)

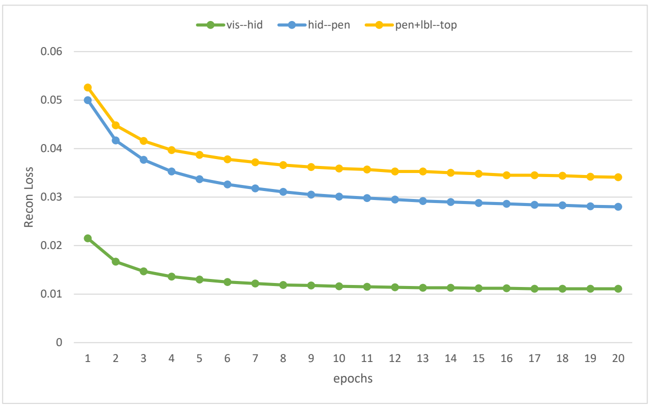

Figure 3: Reconstruction loss over epochs for the three stacked RBMs

When trying to generate numbers we see more like random number-like structures than the numbers we try to produce.

3.3 Supervised fine-tuning of the DBN

One can see in Table 2 that 5 epochs of fine-tuning are improving the networks (5 epochs greedy init) classification performance by around 5%. Somehow, when we increased the number of epochs for the greedy layer-wise pretraining for example to 20, the fine-tuning was not improving the classification performance anymore.

Network	Greedy Training Time (sec)	Test Accuracy (greedy)	Fine-tuning Time (sec)	Test Accuracy (fine-tuned)
784, 500, (500+10), 2000	526	80.80%	670	85.33%
784, (500+10), 2000	445	81.57%	586	83.87%

Table 2: Comparison of standard and smaller network (trained for 5 epochs, greedy and fine-tuning resp.)

The smaller network can reach around the same accuracy after 5 epochs of greedy layer-wise pretraining but performs a little bit worse after fine-tuning for classification.

When looking at the generation capabilities this changes are lot. The performance of the fine-tuned "standard" model is better than the non-fine-tuned one. Here, one can recognise real numbers instead of nearly random structures. For the smaller model the generation is quite weak and producing a lot of zeros and only a small fraction of values higher than zero.

Our best model was able to reach 88% classification accuracy. It was pretrained for 20 epochs and then fine-tuned for 5 more epochs. Strangely, the generation capability of this model seemed quite bad because there was no big difference between generating different numbers.

Because the increased calculation times for the assignments in this lab slowed down our work, we also investigated the impact of the number of training pics on the calculation time. In Table 3 are shown the calculation times and accuracy for reduced numbers of training pics. As you can see the accuracy is strongly decreasing. For this reason we decided to train the network with 60000 pictures.

Training Pics	Accuracy after Greedy-Learning	Accuracy after fine-tuning	Calculation Time [s]
60000	59%	77%	567
40000	52%	70%	401
20000	41%	63%	245

Table 3: Comparison of accuracy and calculation time for full network. Two iterations for greedy learning and fine-tuning.

4 Final remarks

What was nice to see in this assignment, was how the different fine-tuning steps (sometimes only making the network bigger) was improving the performance of the model continuously. We have also seen that the hyperparameters have a huge influence and that they were well chosen in the standard setup. Every time we changed them only a little bit, we could see a drop in performance.

What we have to keep in mind, is that we performed for every experiment only one run because of time reasons. Therefore, there is a random factor involved.