Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 4

Restricted Boltzmann Machines and Deep Belief Nets

Maximilian Auer, Lukas Frösslund and Valdemar Gezelius

October 30, 2020

1 Main objectives and scope of the assignment

The main objectives of this lab was to get familiarised with deep neural network architectures, with a focus on Restricted Boltzmann Machines and Deep Belief Nets. The scope of the assignment included algorithms used in these architectures, such as contrastive divergence, as well as strategies to effectively train these networks, such as greedy layer-wise pretraining.

2 Methods

For all parts of the lab, Python 3 was used together with NumPy for mathematical operations and Matplotlib for plots.

3 Results and discussion

3.1 RBM for recognizing MNIST images

On the question of monitoring the stability, or convergence, in the behaviour of the units, one straightforward approach to do so across iterations is to examine the average reconstruction error, i.e. the squared error between the initial data and the reconstructed data, produced in the model. It's important to note though, that while some trends indicate good convergence (for example a rapid fall in the early learning process and then a slower error decrease), Hinton states clearly that it is a poor measure of the true learning process, because it's not the error function that the implemented contrastive divergence learning is optimizing[1].

Despite the aforementioned flaws of examining the reconstruction error, we decided to use it as a measure of convergence namely because of two reasons. First and foremost, Hinton instructs in the paper to use it, but not trust it completely. Second of all, there is a lack of other quantitative measures. We will later visualize the receptive fields, the probabilities of hidden units and examine

histograms of the weights, which are Hintons alternative recommendations of monitoring the learning.

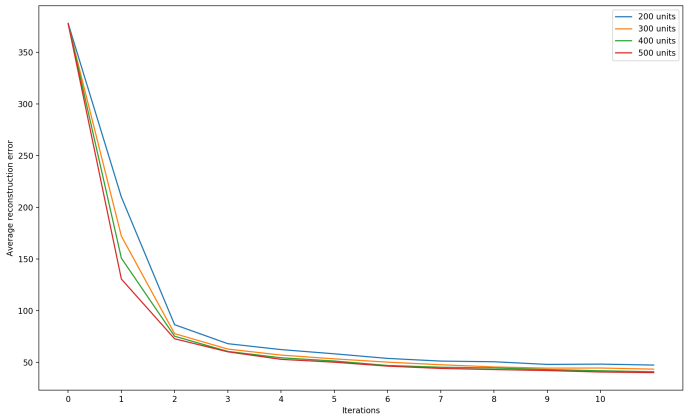

Figure 1: Average reconstruction error across iterations (error calculated every 100th iteration) for different number of hidden units.

We can see in Figure 1 that the reconstruction error decreases more rapidly when using more hidden units, which is to be expected in this case. We can also confirm the trend that Hinton alluded to, that the error would decrease significantly at the start when using the complete training set (which we did here), to then slow down. The fact that our process is in line with this statement strengthens our claim that we are in fact seeing stability and strong convergence in our learning process.

In his guide on training Restricted Boltzmann Machines[1], Hinton describes three effective methods of displaying what is happening during the learning process. The first one is to visualize the weights, visible biases and hidden biases as histograms, before the learning process and then after a few iterations of training.

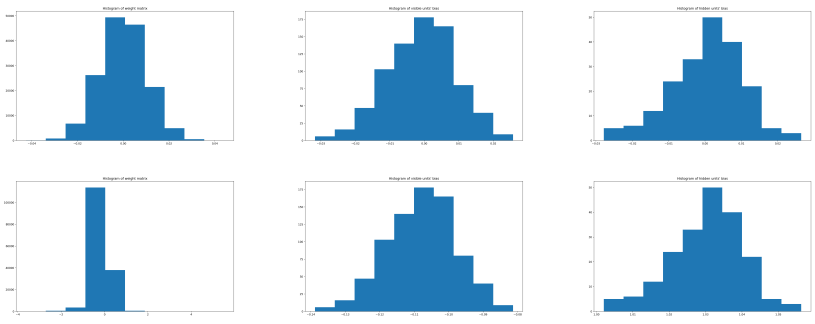

Figure 2: Histograms of weights, visible biases and hidden biases before and after learning.

It's difficult to make any notable implications based on the visualizations of the histograms in Figure 2. We can see that for the visible and hidden biases, the distributions look very similar before (top row) and after learning, with a difference in that their means have deviated. For the weights (left column), we can see the typical gaussian shape at initialization and what we see after learning actually contradicted our initial hypothesis and the literature that we have read. While it does peak around zero, the deviation of the data have become much larger, which is not in line with earlier research. According to[2], the mean absolute magnitude of the weights should decrease significantly in the learning, but in our experiments the reversed happen. Other reports have pointed to that a change in learning rate could alleviate this problem, but experiments with the learning rate did not prove successful in our experiments.

The second method described and proposed by Hinton is to visualize the receptive field for each hidden unit, in an effort to see and understand which features that the units have learned. We chose to visualize the receptive fields before and after learning, to understand the feature extraction process and to further be assured that the convergence has been successful.

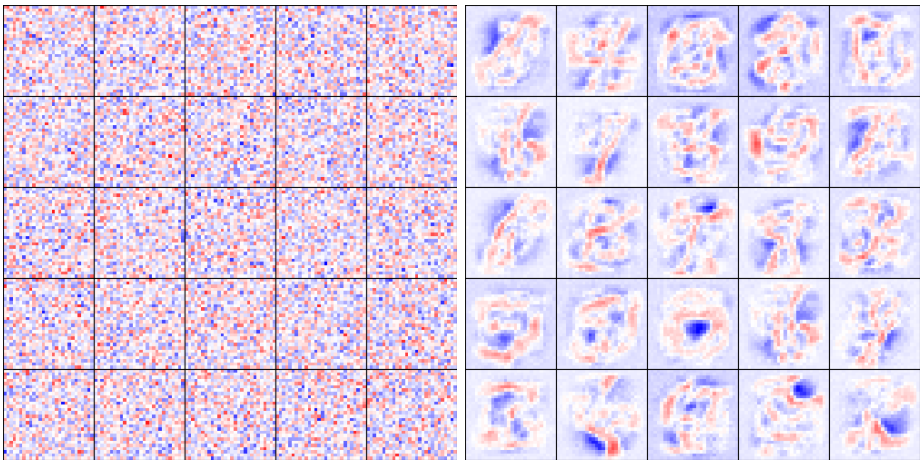

Figure 3: Visualization of receptive fields **Left:** Before training. **Right:** After one epoch of training.

In Figure 3, we see the visualized receptive fields, i.e. the weights connecting hidden to visible units. On the left we have the receptive fields at initialization, and it's sufficiently obvious that the network has not learned and extracted any features here as the patterns look completely random. On the right however, we can see that the receptive fields after training have extracted distinguished features that resembles handwritten digits. What is interesting here is that we can clearly notice that the receptive fields have extracted combinations of features from several digits, rather than a single digit.

Additionally, it is recommended to visualize a two-dimensional grid display of a single arbitrary mini-batch used in the learning process. The vertical axis of the grid represents the different training samples and the horizontal axis is each binary hidden unit, where we visualize the probability of activation by a

gray-scale pixel value between 0 and 1. What we want to see here is uncertainty in the network, with patterns that looks sufficiently random.

Figure 4: Probabilities of activating hidden units in a minibatch of 10 digits. 200 units in hidden layer.

Analyzing Figure 4, we can see that the patterns are sufficiently random and we can confirm that there is uncertainty in the network to a relatively high degree. There are no clear horizontal lines, which is positive because it suggests that no sample in the minibatch activates for an unusually large or small number of units. We do see a couple of vertical lines, which suggests that a very small number of units seems to behave the same regardless of training sample. This is ideally something we want to avoid, but given that there is very few of them, it shouldn't be a significant problem.

As a final analysis of the learning process, we chose to run a small subset of test images through the trained RBM and compare the reconstruction to the input test image. As can be seen in Figure 5 below, the reconstruction is seemingly quite accurate. Small deviations can be spotted, where the recalled reconstruction tends to be more noisy, but overall we can clearly see that the RBM has converged sufficiently well.

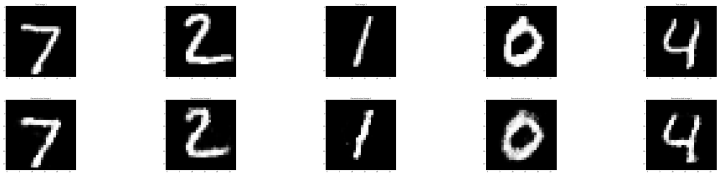

Figure 5: Test images of handwritten digits from the MNIST dataset and their corresponding reconstructions from the trained RBM.

4 Towards deep networks - greedy layer-wise pre-training

We can extend a single hidden-layer network to a "deeper" architecture by stacking several RBMs on top of each other and thus create a so called Deep Belief Network, DBN. This is done by performing greedy layer-wise pretraining, i.e training each RBM separately and then direct the weight-layers between visible- and hidden-layer and use previous RBMs hidden-layer as visible layer for the next RBM. For the first part we used a batchsize of 10 for pretraining.

In Figure 6 we see a graph showing the average reconstruction error from training the first and second layer of our Deep Belief Network. The structure for the network use two RBMs structured 784-500-500 units corresponding to visible-hidden-top. The hidden layer becomes the visible layer for the top layer. The

first layer, visible-hidden, correspond to an ordinary RBM. We can draw the conclusion that the reconstruction error seems to converge to a lower error when we stack RBMs, since the second layer uses the information learned in the first layer, but the difference is not significant.

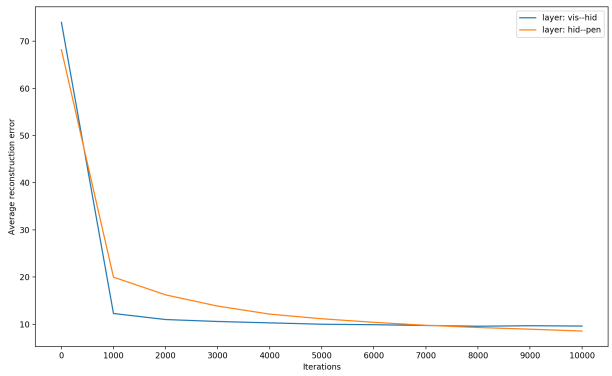

Figure 6: Average reconstruction error per 1000 iterations for first and second layer of the DBN.

Figure 7 shows a test image from the MNIST dataset and the reconstruction made by a RBM and our DBN. We can see that there is no significant difference between the two reconstruction and since it did not differ significantly in the reconstruction error (shown in Figure 6) we can draw the conclusion that for this purpose, there seems to be enough to train a RBM.

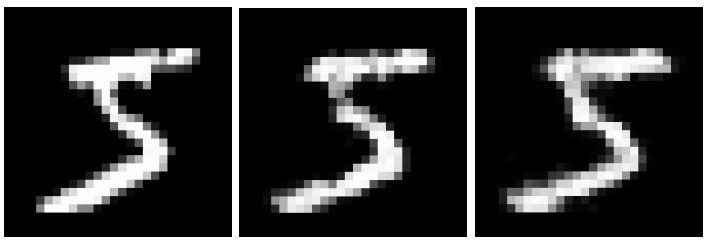

Figure 7: **Left:** Handwritten digit from the MNIST dataset. **Middle:** Reconstructed image using a RBM. **Right:** Reconstructed image using a DBN.

We then extended our DBN to be able to accommodate labels. The structure of the network becomes 784-500-500-2000 units, corresponding to visible-hidden-penultimate-top. In the penultimate layer we concatenate one-hot vectors representing the labels. Figure 8 shows the average training- and test accuracy when predicting labels, using a batchsize of 40. We get a resulting training accuracy of ≈85% and a test accuracy of ≈86%.

The labels are set to an uniform distribution at start and then predicted as the argmax after completing 15 steps of Gibbs sampling (k=1) from the top layer.

Misclassification from our network be caused by similarities in features between the digits in the dataset, e.g a 9 and a 4 are not orthogonal patterns, especially when handwritten.

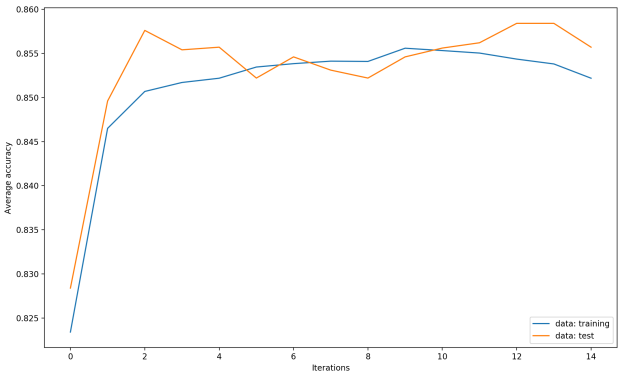

Figure 8: Average prediction accuracy across iterations for the DBN.

For generating data from the networks learned distribution, we fix the labels and perform Gibbs sampling in the top layer. We then use the generated probability to propagate back through the network and finally get a probability distribution for our generated visible layer. If we perform 200 Gibbs sample iterations (k=1), we generate 200 samples corresponding to the current label. Since we start from random samples it takes some iterations to get anything resembling what we are after. We see that batchsize has an effect on the result when training the network, with better results for batchsizes >10.

Since we have not fine-tuned the network, the results are not optimal. Figure 9 shows that for the labels corresponding to 0, 3, 5, 6, 8 and 9, the network does a decent job, but for remaining labels, the network seems to have a hard time distinguishing between features corresponding to current label and features corresponding to a different label. A conclusion could, once again, be that the patters are not orthogonal.

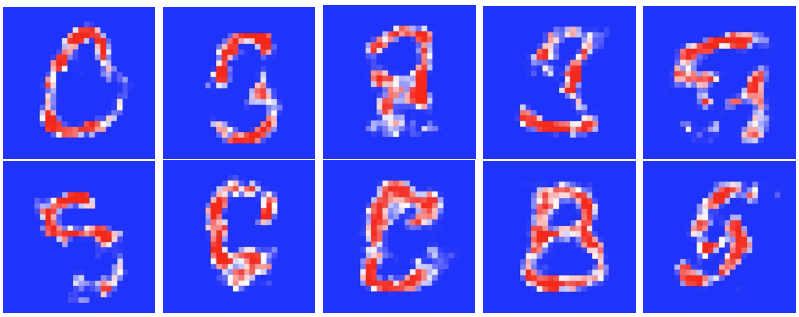

Figure 9: Generated images from labels using a DBN, batchsize=40. **Top, left-right:** 0, 1, 2, 3, 4 **Bottom, left-right:** 5, 6, 7, 8, 9

5 Final remarks

Overall, the lab yielded good insights into several aspects of Restricted Boltzmann Machines and Deep Belief Nets, and granted us with an understanding of both the potential and the limitations of the networks. Writing just parts of the code was convenient in the sense that more effort could be put on the theory and experiments, but it also in a sense hindered understanding because we didn't code everything from scratch as we've done previously.

References

[1] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

[2] Jason Yosinski and Hod Lipson. Visually debugging restricted boltzmann machine training with a 3d example. In *Representation Learning Workshop, 29th International Conference on Machine Learning*, 2012.