Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 4

Restricted Boltzmann Machines and Deep Belief Nets

Ziyuan Wang, Emil Kerakos, Ming Jiang

February 25, 2021

1 Main objectives and scope of the assignment

In this lab our aim was to get familiar with Bolzmann Machines and Deep Belief Nets. More specifically, our major goals in the assignment were to:

- Get a solid understanding for the learning process of RBM:s,
- Get familiar using basic algorithms for unsupervised greedy pretraining of RBM layers and supervised fine-tuning of the resultant DBN,
- See how multi-layer neural network architectures based on RBM layers can be used for classication.
- Study the functionality of DBNs

Due to the limited time and computational resources this was not meant to be an exhaustive investigation of RBM:s and DBN:s and we only used image data. The scope of this particular assignment is well-structured and fulfilled for our capacity. And the limitation for the given project is that some tasks are similar but are separated into different sections

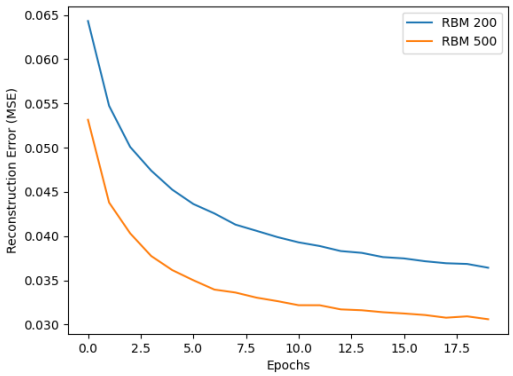
2 Methods

The coding language for our task is built upon Python 3.6 due to the overall universality for dependencies. For the entire project, we conduct the code based on NumPy 1.19.2 and Matplotlib 3.3.3 for visulization.

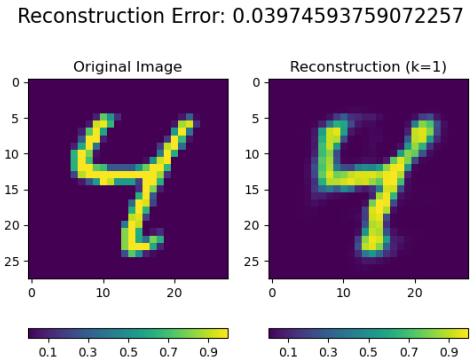
3 Results and discussion

3.1 RBM for recognising MNIST images

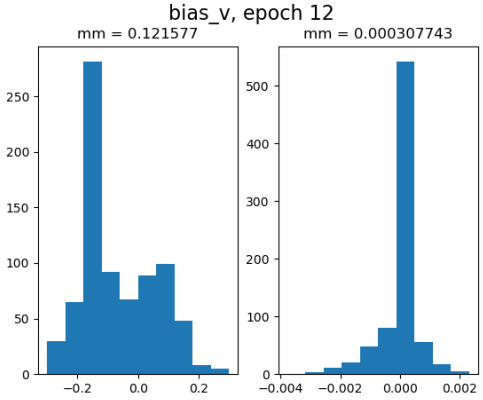
In order to practically familiarize ourselves with the Restricted Boltzmann Machine (RBM) before moving on to the more effective and interesting challenge of stacking RBM:s into a Deep Belief Network we did a simple experiment with images from the MNIST dataset (images of handwritten numbers 0-9) where we trained a RBM with 200 hidden nodes and one with 500 hidden node on 60



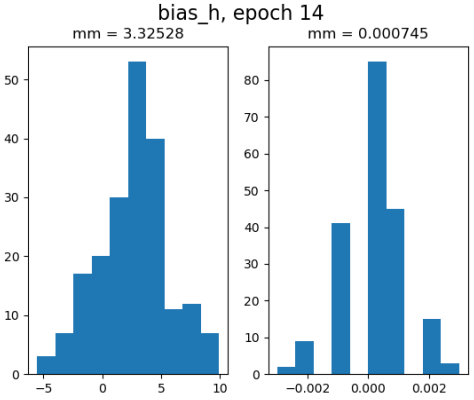
(a) Training and reconstruction loss for an RBM with 200 hidden nodes and one with 500 hidden nodes



(b) Example of a reconstruction after only 3 epochs for the RBM200



(a)



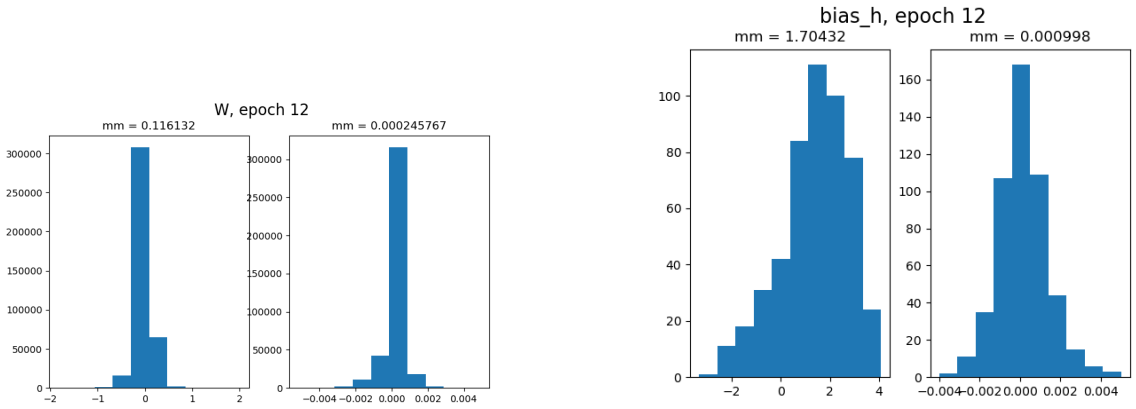
(b)

Figure 2: Less good examples of histograms for RBM parameters (left histograms) and their update size (right histograms) in the middle of training (12 and 14 epochs in respectively) for (a) the visible-bias (b) the hidden-biases.

000 images, both had 784 visible nodes as the images where 28x28 pixels. We ran both RBM:s with the contrastive divergence algorithm (CD_1) with a minibatch size of 20 for 20 epochs. Although the RBM models a probability density in an unsupervised manner, to get some approximate quantitative measure of how well the algorithm is doing we computed and visualized the reconstruction error (defined as mean square error) between each of the 20 epochs.

As we can see in fig 1a, the RBM with more hidden units has a lower reconstruction error over all epochs of training and converges to a lower level, evaluated on the training data. However, on test data the RBM200 performed better but only marginally: 0.3634 vs 0.3649. This is a bit strange and we're fully confident as to why at this moment.

It might be due to more nodes inducing overfitting but it might also be a limitation of the measurement of reconstruction error. As described in [1], to monitor convergence during training there are more considerations to take into account than solely reconstruction error as reconstruction error doesn't follow the true objective function that CD_1 is improving and thus a decreasing reconstruction error does not necessarily mean that the model is getting better [1]. However, even if it's not sufficient it can nevertheless be used as a necessary condition for convergence: the reconstruction should consistently and rapidly fall initially and then more slowly [1], and this is very much inline with what we observe in fig 1a for both models. In 1b we see an example reconstruction. One additional,



(a) (b)

Figure 3: Good examples of histograms for RBM parameters (left histograms) and their update size (right histograms) in the middle of training (12 epochs in) for (a) the weight-matrix and (b) the hidden-biases.

recommended way to monitor learning is to at regular epoch intervals visualize all model parameters as well and their updates using histograms [1], making sure that the mean absolute magnitudes of the updates are 10^{-4} the size of the params and that the distributions are roughly gaussian [4]. These types of plots can be seen in fig 3 and fig 2 for our learning example where in 3 we see one such visualizations where the distributions and updates for matrix W and bias-vector $\vec{b_h}$ for the hidden units do satisfy these properties pretty well. In fig 2, the distributions are not as good as there are very "tall spikes" in the distributions that otherwise are gaussian-like. The updates are still in the right order of magnitude though and the "empty regions" in (b) are still inline with example plots given in [4].

Beyond visualizing reconstructions, one way to get insight into the RBM is to visualize the weights associated with individual hidden units [1]. In this way we can see which pixels increase and decrease the probability of a hidden node firing: in other words which features have the hidden nodes learnt. As can be seen in 5x5 grid in fig 5 we can see some interesting things. Each gridcell is a hidden neuron and for visualization purposes we only display 25 here. Red pixels denote pixels that increase the probability that the the hidden node corresponding to the grid cell is activated if the pixel values have a high level of grey, blue pixels are the opposite. For instance for the first image on the second row we see a dark blue spot at the top, a lighter curved red horizontal line just below it and then one at the bottom. It thus seems that a 3, a 5 and a 7 would increase the probability of it being activated a lot. The fifth picture on the second row seems to be a mix of a 3 and a 5. The hidden unit corresponding to the picture on the last row has all weights quite small except for the middle pixels: it is thus activated if there are a high level of gray for pixels in the middle and it doesn't care much about the rest. Thus the middle portion of an 8 or a 3 should activate it.

3.2 DBN with greedy layer-wise pretraining

In this section, we first train greedily two RBMs in the stack with CD algorithm. We plot the reconstruction losses for both RBMs as shown in Fig. 7 with iterations of 15 and 20. Next, we add another RBM layer on the top of the previous RBMs stack. The hidden outputs in the second RBM together with 10 labels dataset are the inputs of this top layer 510-2000 RBM. We use the pre-trained DBN for image recognition and with 15 iterations training get an accuracy of **87.43%** on training set, while **87.85%** on test set.

Finally, we use the pre-trained DBN as a generative model. We try to generate 200 samples of each

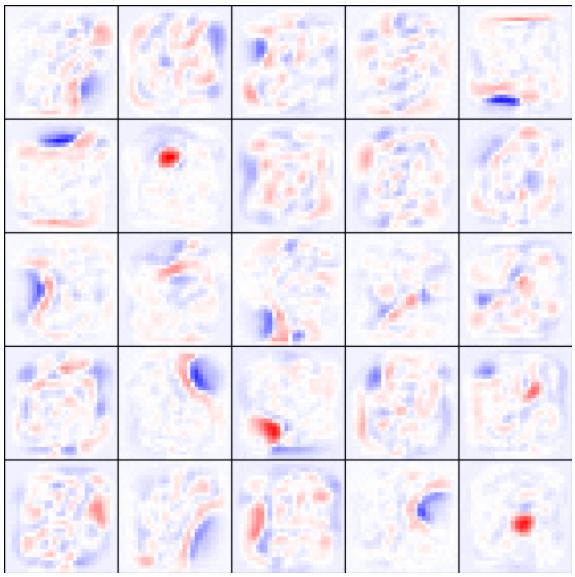
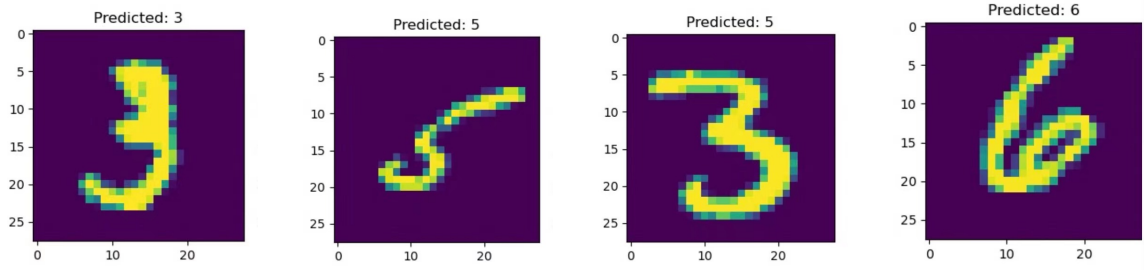


Figure 4

Figure 5: Visualization of the weights after 20 iterations of training for the RBM200.

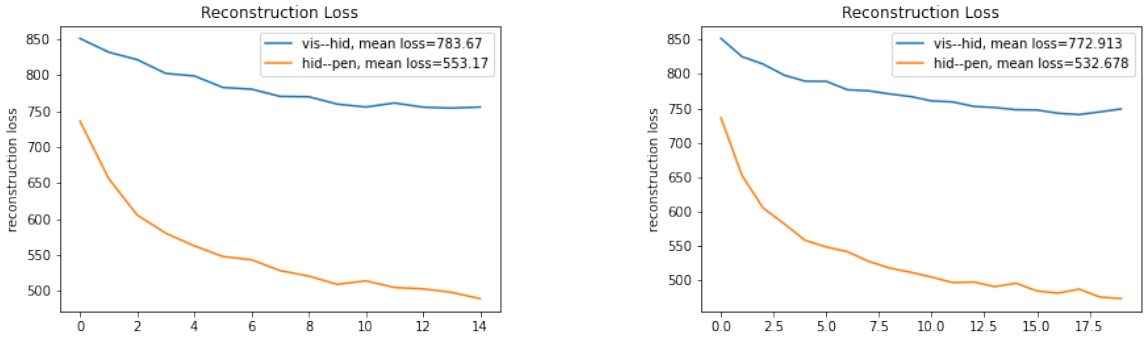


(a) (b) (c) (d)

Figure 6: Recognition Results Examples of Pre-trained DBN(*iter=15*)

digits and Fig. 8b and Fig. 8d shows the examples of digits *3* and *9*.

- Reconstruction loss of the second RBM is less than the first one. After running the second RBM, we actually get data with less noise.
- With the slight increase of iterations, the accuracy has a improvement, which means that we get a more well-pretrained model.
- The reconstruction loss on test set is lower. It might due to that we are actually doing the unsupervised learning the error of training set and test set can not be understood as in the supervised learning. In this task, the training set is 6 times larger than test set. Besides, the better performance on unseen data than the seen data proves that our model is not overfitting. However, it might also have the risk of not being fully trained.
- Fig. 6 show some examples of the recognition. We can see it does have a good performance even involving some weird digits(Fig. 6a). However, some recognition errors occur in the vague and similar digits such as Fig. 6c.
- Without fine-tuning, the directed parts of DBN are not even shown, because *weight_h_to_v* and *weight_v_to_h* are just the transpose of each other. As a result, the DBN now might have a good performance on the discriminate task, but not does good enough on generative task[2].



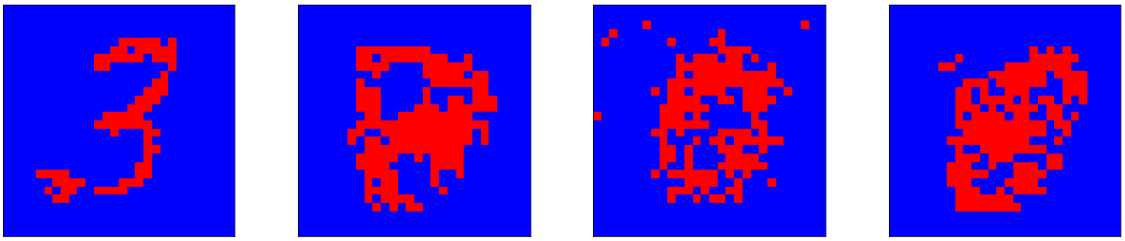
(a) Iterations: 15

(b) Iterations: 20

Figure 7: Reconstruction Loss of two RBMs in DBN with Different Training Iterations

3.3 DBN with wake-sleep fine-tuning

In this section, we took the model into next level of accuracy with the technique of wake-sleep fine-tuning. Essentially, the wake-sleep algorithm is to treat the transpose direct weights as two sets of weight, and then adjusting the weight using back propagation. Cognitive weights are used to drive neurons from bottom to top, and the binary states of neurons in adjacent layers can be used to train and generate weights. The top-to-bottom generative connection is used to drive the network to generate images based on the generative model[2]. After altering the code for this task, we get the result of $training\_recognise\_error$: 91.63%, $testing\_recognise\_error$: 91.71%, which is averagely 4% higher than the greedy algorithm mentioned Section 3.2. And we could spot the differences lies in generative capacity with the generated images with given label, the vast differences of visualisation could be seen in Fig 8a and Fig 8b ,however, we do experience some level of vagueness of visualisation which could be find in Fig 8c and Fig 8d.



(a) Fine-Tuning Generation of Number 3

(b) Greedy Training Generation of Number 3

(c) Fine-Tuning Generation of Number 9

(d) Greedy Training Generation of Number 9

Figure 8: Difference between fine-tuning and greedy DBNs

As for the theoretical question about stacking RBMs, we think this section is a decent place for explaining it. In a DBN the first two layers form an RBM (an undirected graphical model), then the subsequent layers form a directed generative model for inferring process. And the top of the network remain undirect to generate, as well as the direct model is for learning process for given network. And when from top to bottom, we need to use the directed connections to transform the signals or features from undirected connections to inputs patterns[3].

4 Final remarks

In this lab session, we have learnt the Bolzmann Machine and Deep Belief Nets from a methodology perspective as well as a practical perspective. The overall experience is significantly beneficial for understanding the foundation of Graphical Model which is a much more similar simulation between computer and the human brain.

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] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[3] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Artificial intelligence and statistics*, pages 448–455. PMLR, 2009.

[4] 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.