Short report - Lab assignment 4

Restricted Boltzmann Machines and Deep Belief Nets

Adrian Campoy Rodriguez, Nimara Doumitrou-Daniil and Gustavo T. D. Beck

February 28, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were: (1) Understand and explain the main features of RBM (2) Build a basic RBM network trained using unsupervised greedy pretraining and supervised fine-tuning of the resulting DBN after stacking several RBM (3) Apply the created networks for classification and generation problems.
In general, during this assignment, our main intention was to get familiar with the key concepts of Deep Neural Networks and, in particular, Restricted Boltzmann Machines and Deel Belief Nets.

2 Methods

During this project the programming language Python 3 was used in three different IDEs such as Pycharm, VSCode and Spyder. Some particular toolboxes were used for specific tasks. For instance, matplotlib was used as an additional support to visualize the images and the generated patterns. Finally, the GitHub repository was used to store and manage different versions and pieces of code.

3 Results and discussion

Boltzmann Machines augment the architecture of the Hopfield Networks we experimented with in the previous lab, by adding extra **hidden nodes**. They also get rid of the deterministic outputs of the nodes (based on the sign of their inputs). In the case of binary inputs, they instead follow a bernoulli distribution that is greatly affected by the temperature of the network. Even though such networks are complete, in the sense that given enough time (temperature decreases with time), the network will successfully approximate the desired distribution, their learning is fairly slow due to the numerous existing connections. For this reason, the Restricted Boltzmann Machine aims to find the sweetspot between Boltzmann Machines and Hopefield Newtorks, by removing the mutual connections within the hidden nodes and the visible nodes.

3.1 RBM for recognising MNIST Dataset

At its core, an RBM network consists of a visible layer v, a hidden layer h, a matrix W connecting them and two bias vectors b_v and b_h. The general update rule after k Gibbs samplings for the weights is $\Delta w_{i,j} \propto < h_j \hat{v}_i >^{t=0} - < \hat{h}_j \hat{v}_i >^{t=k}$. In fact, Hinton et. al. showed that this update is equivalent to minimizing $KL(P_0^{data}||P_\infty^{model}) - KL(P_k^{model}||P_\infty^{model})$, P_0^{data} is the data's distribution, P_∞^{model} is the model's distribution at equilibrium (after sufficient Gibbs sampling), P_k^{model} is the model's distribution after k Gibbs samples. Gibbs Sampling always reduces the Kullback-Leibler divergence between P_k^{model} and P_∞^{model} and $KL(P_k^{model}||P_\infty^{model})$ is lower bounded by zero.

Thus, we are guaranteed that after enough iterations k_{opt}, $KL(P^{model}_{k_{opt}}||P^{model}_{\infty}) = 0$, ensuring convergence to the true underlying equilibrium distribution of the model.

Equipped with this knowledge, we attempted to reconstruct 28×28 pixelated images of digits. The MNIST dataset consists of evenly distributed images of handwritten $0 - 9$ digits. We trained our network by inputting images in our visible layer v^0, and performing Gibbs sampling of $k = 1$ and updating our parameters based on the generalized weight update rule (in the formula, variables with hats correspond to probabilities). Note that we closely followed Hinton's illuminating tutorial paper, utilising a learning rate of $\eta = 0.01$ and momentum of $\alpha = 0.7$. In our experiments, we examined the effect that the number of epochs and the size of the hidden layers have on our reconstruction loss function $||v^0 - v^1||_2$.

Firstly, the training process (Contrastive Divergence) was iterated for a varying number of epochs (from 10 to 20) using minibatches of size 20. The idea was to observe convergence after a certain amount of iterations.

The method used to check whether units were converging was average reconstruction loss. Reconstruction loss was calculated as the norm between the values of the visible layer and the values of the same visible layer after applying Gibbs sampling. It is worth mentioning that the inputs from the original images in the visible layer are a value between 0 and 1 whereas the values after the Gibbs sampling will be a binary value (0 or 1).

Secondly, the number of hidden units used was changed from 500 down to 200 to observe the effect that this may have on the average reconstruction loss.

The results are showed in figures 1, 2 and 3

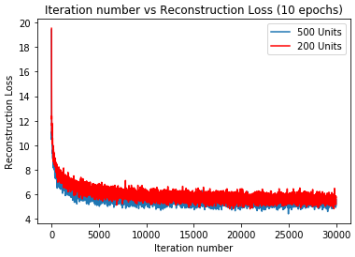

Figure 1: Reconstruction Loss. 10 Epochs.

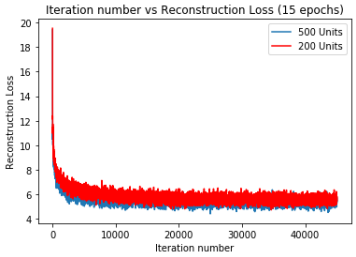

Figure 2: Reconstruction Loss. 15 Epochs

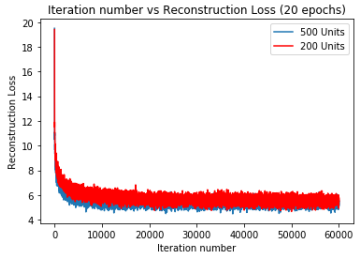

Figure 3: Reconstruction Loss. 20 Epochs

From the previous figures, it can be observed that the network converges approximately after 7500 iterations (2.5 epochs). Moreover, although with 200 and 500 the networks have similar reconstruction loss values, the values in the network with 500 units are slightly lower than those for the network with 200 units. This effect is expected since having more units should allow a better representation of the input images and, as a consequence, a lower reconstruction loss.

The reconstruction loss over 20 epochs was also computed for the test data set to assess the generalisation capabilities of the network. The reconstruction loss was very similar between the train and the test. Nonetheless, as expected, test reconstruction loss was higher than that of the training data set (see figure 4).

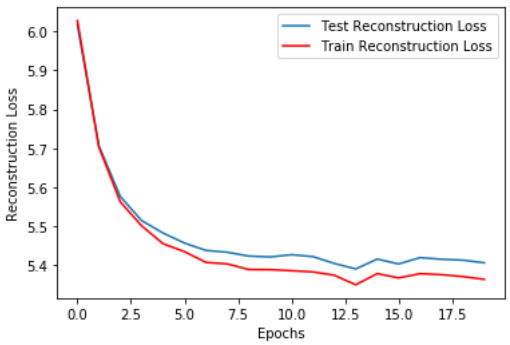

Figure 4: Reconstruction Loss for test and training data sets.

Finally, considering that the training has been conducted without the use of labels, the evaluation fully relies on examining the reconstruction of images as well as the receptive fields created.

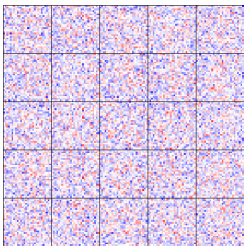

Figure 5: Receptive Field. 0 Iterations.

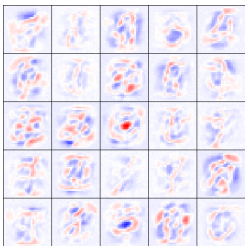

Figure 6: Receptive Field. 5000 Iterations.

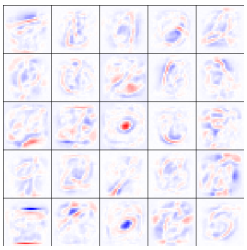

Figure 7: Receptive Field. 60000 Iterations.

From figures 5, 6 and 7 it can be seen that with 0 iterations the regions or features of the input space that hidden units are *looking at* are random since no training has been done yet. As the number of iterations is increased it can be observed how each unit *looks* at different features from the input space. Certain patterns or numbers can indeed be seen in the receptive fields from figures 6 and 7.

3.2 Greedy Layer Wise Pretraining

One can stack multiple RBMs on top of each other (the hidden layer of RBM^i constitutes the visible layer of RBM^{i+1}) in order to build a Deep Belief Network (Deep Boltzmann Machine). We can then train its parameters by adopting a greedy approach where we iteratively train the $i-th$ layer by fixing the previous layers' parameters, treating their hidden outputs as visible data (note that we are inputting the probabilities $\hat{h}$) and performing the same training we used in the previous section. This algorithm is "greedy", as weight W^i is never retrained. We utilise this approach both for image reconstruction (784-500-500) and for image recognition (784-500-(500 + 10)-2000), see figures 8 and 9, which can be observed that due to the batch training there is a fluctuation while learning due to the different batches, while in testing we test we the same test over and over at each iteration.

Note that in this architecture, the lower layers have unidirectional Generative and Discriminative Parameters. Given a certain input (image for discrimination, label for generation), the network only processes the information in these layers **once**, unlike in the top layer where one can perform multiple Gibbs Samplings. Generative and Discriminative parameters are no longer tied (one is the transpose of the other) and are used in sleep and wake phase, respectively, in the Bottom Up Algorithm (see next section). The top Layer acts as the networkâs associative memory: Bidirectional weights are equivalent to infinitely many higher layers with tied weights (similar to Hopfield Networks).

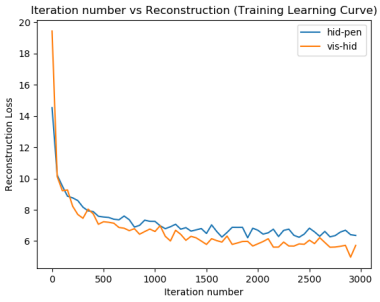

Figure 8: Reconstructions Losses for the training data for both RBMs in the stack (784-500-500).

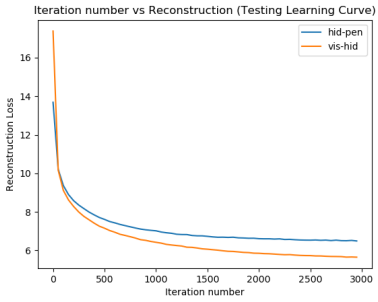

Figure 9: Reconstructions Losses for the testing data for both RBMs in the stack (784-500-500).

In terms of the accuracy of the recognition on the top layer (see table 1) the architecture proved to perform well to discriminate the visible image and retrieve the correct labels of this image:

Belief Network architecture	Accuracy greedy-learning
728-500-(10+500)-2000	85.83

Table 1: Accuracy in testing with greedy-learning method.

We also generated images, by inputting an *average training image* and clamping an one-hot representation of our desired digit (label), performing multiple ($\sim$ 800) Gibbs Samples at the top layer, before propagating our information downwards and outputing an image. Some of our promising results are summarised in the following figure:

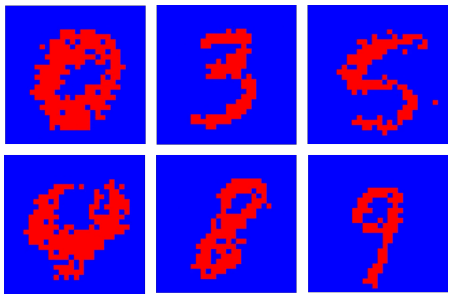

Figure 10: Generated digits (0,3,5,6,8,9) after greedy-learning

The generated images in comparison to the expected original images proved to be reasonable. In most cases the the generated images represented the principal characteristics of their respective original images, in the sense that a human could easily distinguish which number they were representing. However, in some cases, such as the 6th digit, the generated image didn't converge to any specific digit. In addition, it is worth mentioning that the pixels in the generation after each Gibbs sampling weren't stable, which can be seen in figure 10 where the images pixels have some flaws.

If we analyse the accuracy of the model and the generated images, one could expect better images generations, since the accuracy on the recognition was above 80%. However, discriminant models that rely on the distributions $P(y|x)$ are easier than generative models, where the distribution is $P(x, y)$.

If we decided to enhance our model with greedy-learning, some factors play an important role: increase number of Gibbs sampling, increase the number of layers and number of epochs on training the training samples.

3.3 Supervised Fine-Tuning

Albeit the results obtained in the previous section seem quite promising, we nevertheless would wish to try and enhance our training, by attempting to update our parameters W^i in a way that takes into account the whole network's information (unlike the greedy approach discussed in the previous section). We thus implemented the Up-Down Algorithm. We ran said fine tuning for one epoch (after the greedy layer pretraining), utilising 20 Gibbs Samples in the top layer. This further improved our network's accuracy and generation capabilities.

Model/metrics	Accuracy (greedy/fine tuned)	Training Time (greedy/fine tuned)
728-500-(10+500)-2000	85.83 % /88.60 %	213.71 / 350.9
728-(10+500)-2000	88.41 % / 90.6 %	171.2 / 329.3

Table 2: Accuracy and Time metrics for our models with and without fine tuning. Test accuracy is displayed. Training time corresponds to the time (in seconds) required for one epoch during training.

Overall, the results are reasonable. Fine tuning increased the accuracy of both models, requiring considerably more training time per epoch. Furthermore, the simpler model required less time to train. Interestingly, the simpler model exhibited increased performance as a discriminator (classifier). We reason this is likely due to the relatively small amount of training epochs (20 for the greedy), which is not enough to achieve stability and convergence for the more complex model. However, the simpler model struggled as a generative model, showcasing considerably worse images. In fact, without fine tuning, the 728-(10+500)-2000 architecture only managed to generate "1" images. Once more this is expected, as the underlying Generative (joint) distribution $P(x, y)$ is far more complex than the Discriminative $P(y|x)$.

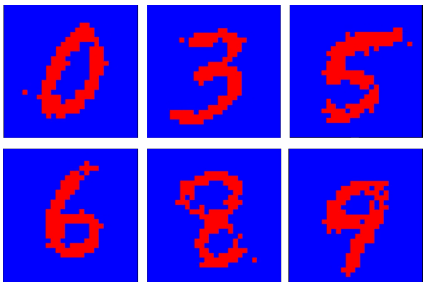

Figure 11: Enhanced generated digits after wake-sleep learning for three layer network. Same digits as Figure 10 are represented for comparison (0,3,5,6,8,9).

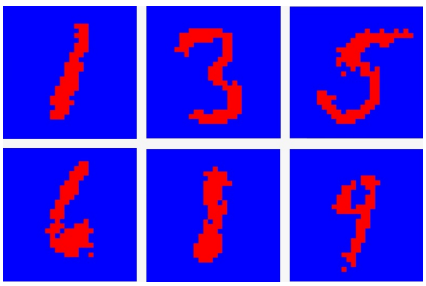

Figure 12: Enhanced generated digits after wake-sleep learning for shallow two layer network. Same digits as Figure 10 are represented for comparison (0,3,5,6,8,9).

The shallow network struggles in generating images of "0" and "8", confusing them with those of "1". Overall, as generative models, we see that (performance wise) greedy (shallow) $<$ greedy(deep) $<$ fine tuned (shallow) $<$ fine tuned (deep).

4 Final remarks

This assignment helped us get familiar with the key aspects of Deep Neural Networks, starting from RBM and stacking them together to create DBNs. It was well structured considering that, given the complexity of these networks, the assignment was proposed in such a way that allowed to start from the basics, building step by step to the most complex tasks and networks. It was very interesting to get familiar both with recognition using DNN and, especially, with generative models taking into account that so far most of our training has been focused on recognition/classification and regression.

It would have been appreciated not only to apply DNN to images but also to other types of data in order to get comfortable with different types of inputs, outputs and how to represent these. Again, as in previous assignments, it would have been interesting to apply some pre-processing techniques to the given images since real-life applications will require these skills.