Artificial Neural Networks and Deep Architectures,

DD2437

Lab assignment 4

Deep neural network architectures with restricted Boltzmann machines and auto-encoders

Raghunath Vairamuthu (rvai@kth.se)
Velisarios Miloulis (miloulis@kth.se)
Xing Guan xguan@kth.se

Our setup:

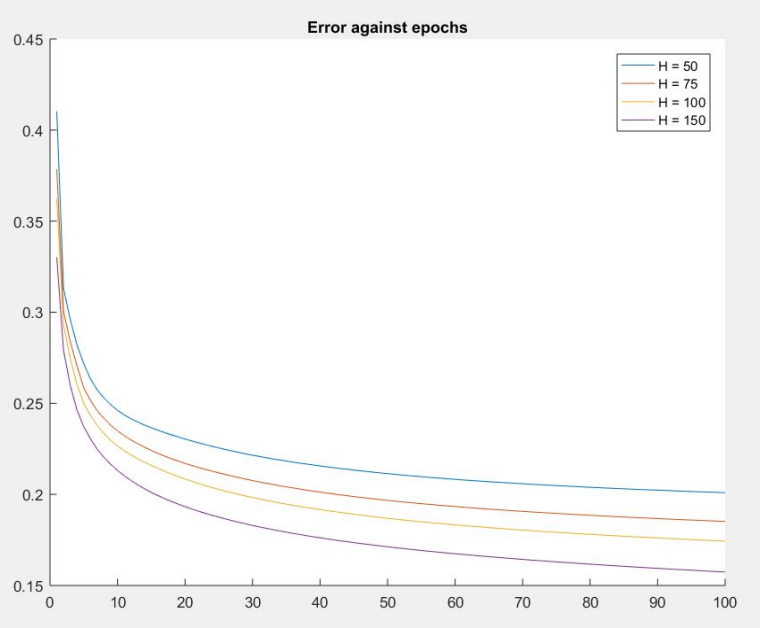
We relied on Deep Neural Network Matlab toolbox by M. Tanaka, for the first part of our work and for part 2, we created our own functions in python for Deep belief networks. The instruction from the lab manual were followed and we discuss our observations and inferences below with appropriate visualsations.

3.1 RBM and Auto-Encoder

RBM:
For each image, the mean error between the original input and the reconstructed input was calculated. Total rmse error on the dataset for every epoch was then obtained and plotted as a function of the epochs after the training was completed. Different number of hidden nodes (50, 75, 100, 150) were used and the errors were compared.

The weight matrix was initialized to random values and the initial bias wa set to 0. The learning parameters are as follows:

Learning rate = 0.1, Number of batches = 8, Max no. of epochs = 100



It can be seen that even though convergence rate for different number of hidden nodes is more or less the same, the RBM with higher number of hidden nodes is able to achieve better performance.

Further observations:

- Larger batch size gives less satisfying result and opposite the error decreases with small number of batch such as 50, 20.
- It is a tradeoff between computation and the batch size. Since it costs more to compute the smaller batches even though we get better results with smaller batches.

- With more hidden nodes the result gets better but computation time increases as well i.e increasing from 100 to 150 nodes, the time increase from estimated 0.7 second to 0.85 second each epoch.

For the following exercise, the maximum number of iterations used was 70.

Reconstruct digit number(index from the train target):

- 0(12), 1(3), 2(9), 3(16), 4(7), 5(4), 6(1), 7(5), 8(32), 9(10).

Original 0 - 9

Reconstructed 0 - 9 (Number of hidden nodes = 100)

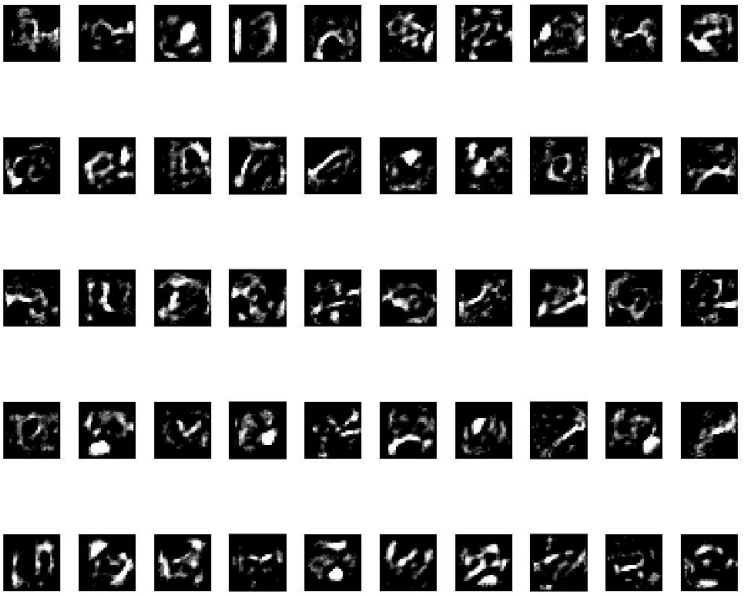
We can see that the images were reconstructed pretty well for the above configurations. To observe how the weights are for different configurations, we plotted them for different node configurations, 50 and 100.

Appearances of weight matrix:

50 nodes

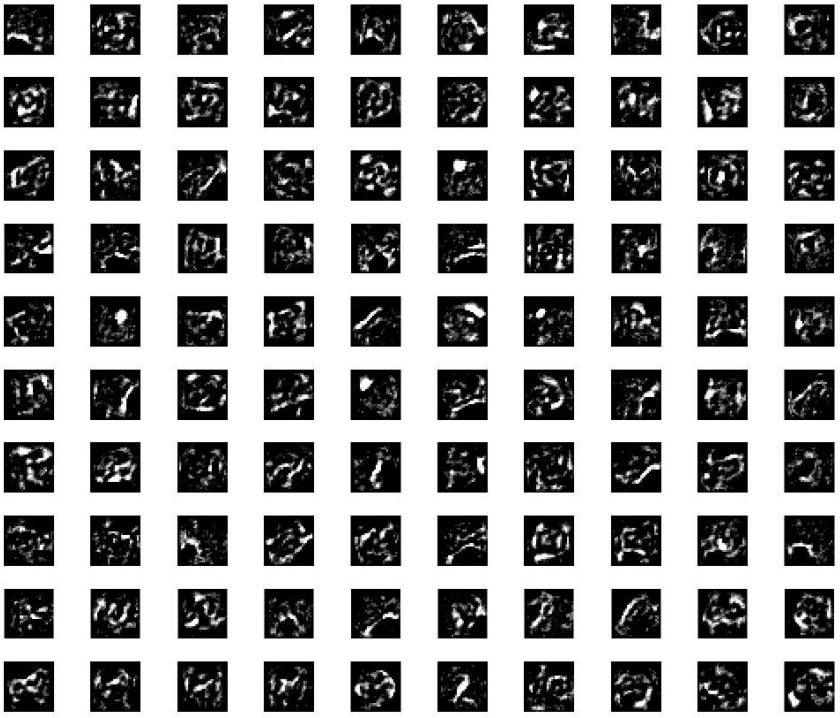
- Each picture represent the weight that is transformed into a 28x28 picture.
- Each weight image is a representation of a particular feature the the network has learnt from the input images

.



100 nodes

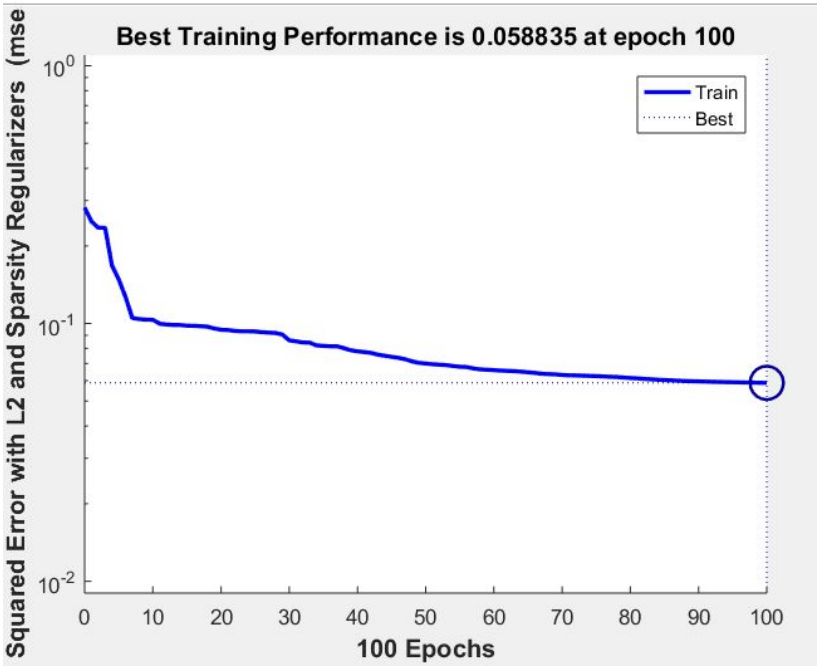
- In a 100 node network, more features are extracted from the input images. This in general increases the training accuracy, even though it may not be ideal to have very high number of node due to high computation cost and loss in ability to generalise future inputs.



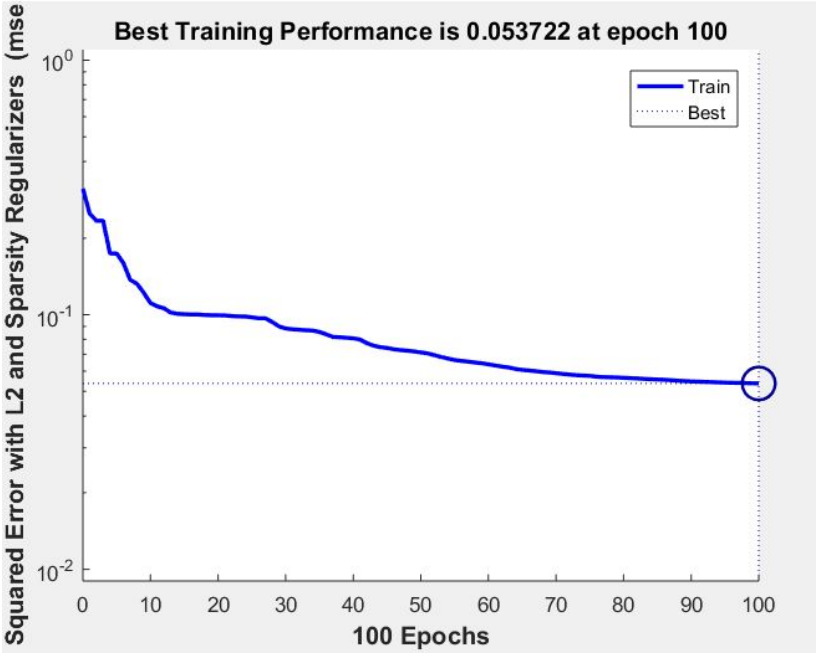
“Translate many low-level features (e.g. user reviews or image pixels) to the compressed high-level representation - but now weights are learned only from neurons that are spatially close to each other.”

Autoencoder:
Experiments similar to RBM was performed with Autoencoder with the same training set and the observations were noted.

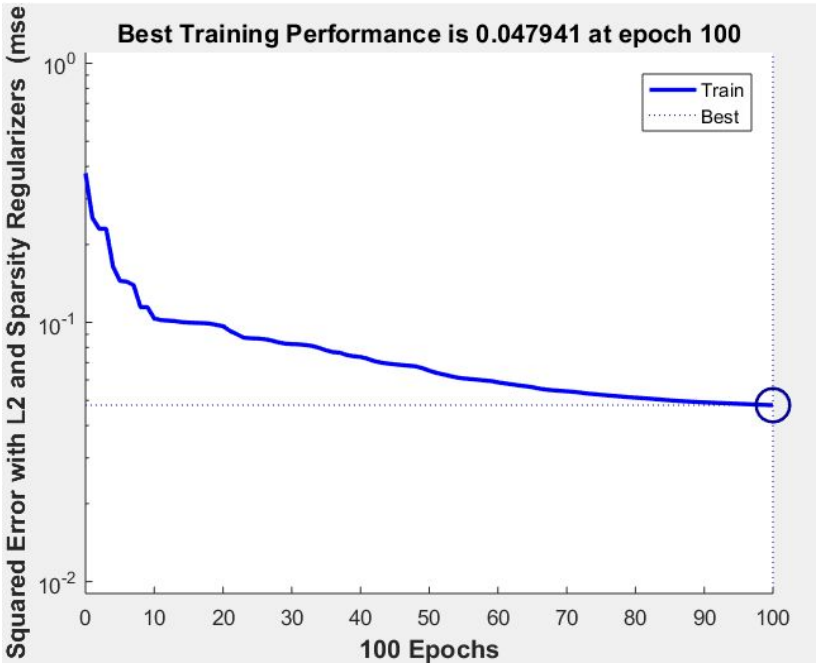
Error versus epohs for different hidden layer configurations:
H = 25



H = 50



H = 100



- The result for autoencoder is similar in context of number of nodes. The more nodes we add the lowerer error it is. It can explained as more feature can be captured from the input data.
- Autoencoders are easier to train and use gradient descent to update the weights, layerwise.
- In the context of image denoising, autoencoder outperform the Gaussian-Bernoulli deep Boltzmann machine (GDBM) with the same number of hidden layers.

Reconstructed images from auto-encoder:

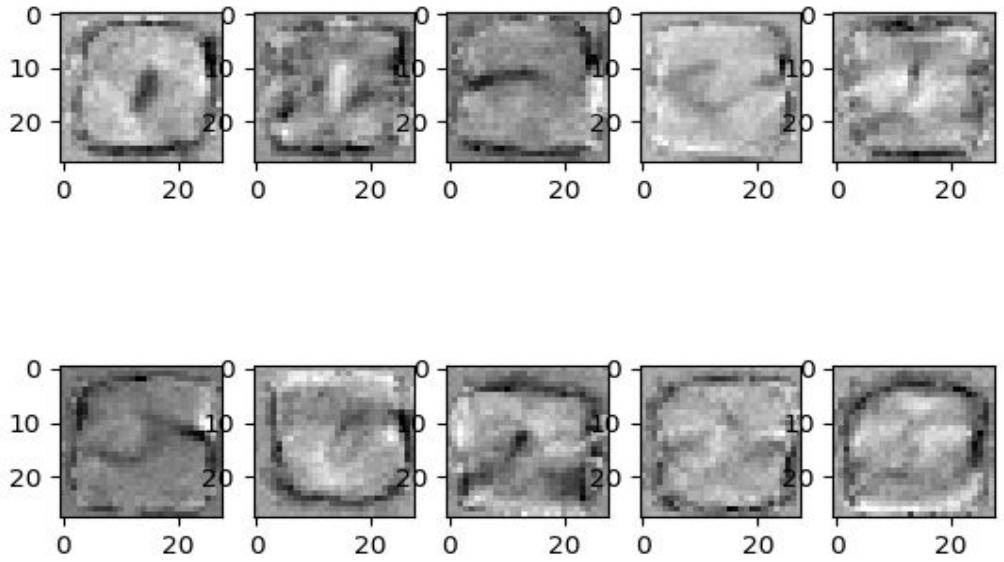
Comparing it with the reconstructions from RBM,

We see that performance of RBM is better over that of an autoencoder. The training error was a bit less in the case of RBM but they did not differ by much.

3.2 DBN and stacked autoencoders for MNIST digit classification

Using the implementations from the previous part, the single-hidden layer network was extended to get deeper architectures. The choice of our size of hidden layer was 150,100, 75 for first, second, and third layers respectively.

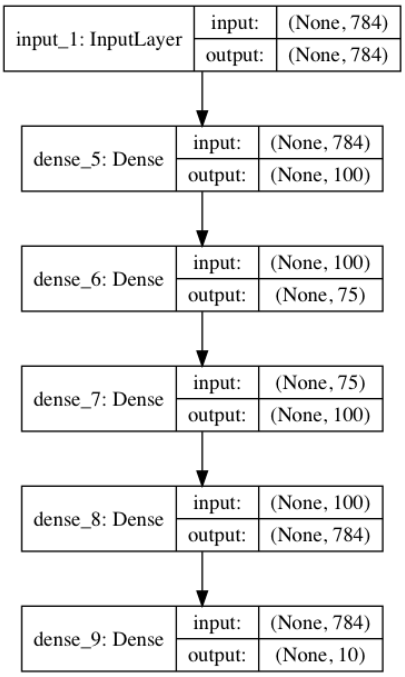
From raw input without hidden layer configuration (Rough idea of hidden weights)

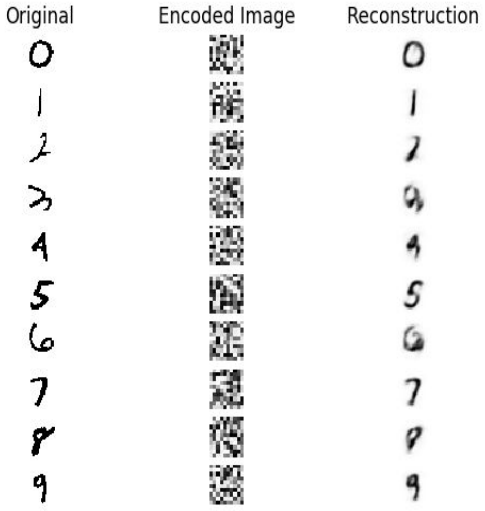


- The image present the weights from non hidden layer network using softmax and cross-entropy.
- We can observe that each image represent a “digit”, some “digit” such as 0, 1, 2, 3, 5, can be recognize as the digits.
- The result is obtained after 20 epochs of training.
- The accuracy is up to 92.69%

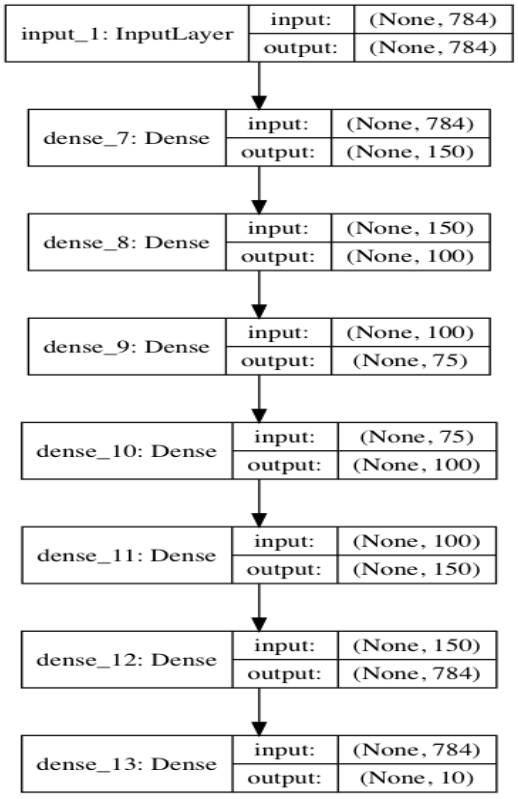
Stacked Auto-encoder:
The image to the left represent the topology of the network. It goes from raw input(784) to 100 and compress to 75 then expand back to 100 and 784. The 784 is the representation of the input which the stacked autoencoder have learned. Using this representation, we can make a predictable model.

2 layer auto-encoder [100, 75]





3 layer auto-encoder [150, 125, 100]



Comparing classification performance obtained with different number of hidden layers for different networks

Training Accuracy

No. layers\ Network type [Hidden layer config]	DBN (acc.)	DBN with backprop (acc.)
1 hidden layer	0.943	0.922
2 hidden layers	0.919	0.917
3 hidden layers [150 100 75]	0.885	0.918
3 hidden layers [150 100 50]	0.856	0.916

Testing Accuracy

No. layers\ Network type	DBN (acc.)	Stacked auto-encoder(acc.)	DBN with backprop (acc.)
1 hidden layer	0.923	0.762	0.910
2 hidden layers	0.910	0.758	0.910
3 hidden layers	0.859	0.759	0.907

- The hidden layer configurations are [150 100 75] unless otherwise specified.
- We observe that DBN have much higher accuracy when we have same settings such as learning rate, epochs and batch.
- The single layer has the best result which was a bit counterintuitive. This is probably because wit more number of layers, we may need to increase allowed number of training epochs. In our experiment, we kept the maximum number of training epochs to be fixed for all networks. We came to this conclusion because the training error also increased with number of layers for the same number of epochs.
- The result differs a lot between DBN and Stacked autoencoder. We can suspect that in the pretraining part, a lot of feature is lost during the compression/when the number nodes of each layer decreases.
- The accuracy drops with increasing number of hidden layers for all networks except for autoencoders where it stays more or less the same.
- The performance for DBN with backpropagation and the accuracy does not vary much with the increase in the number of hidden layers.
- DBN with backpropagation achieved the best results overall.