Artificial Neural Networks and Deep Architectures

Written by: Oguzhan Ugur, Shadman Ahmed, Andreas Yokobori Sävö
E-mail: Oguzhanu@kth.se, shadmana@kth.se, ays@kth.se

March 25, 2018

Introduction

The main parts of this assignment was to get a deeper understanding of deep network architectures. The main points were reconstruction and classification of images using the MNIST dataset consisting of handwritten images of numbers between 0 to 9 in 28x28 pixels. This lab focused on mainly 2 types of DNN architectures: restricted Boltzmann machine and Autoencoder. The networks were implemented by using Deep Neural Network MATLAB toolbox by M. Tanaka.

The given data was normalized and divided into a training and a testing set. In reality however there should also be a validation set to validate the model of choice. This could have been done by splitting the training data into two sets and using one of them for training and the other for validation. However, as this assignment was just an introduction to the concept of deeper networks, this was not employed.

RBM

The Restricted-Boltzmann-Machine(RBM) consists of a visible and a hidden layer with connections between the units from each layer. The visible units are not connected to each other and the same goes for the hidden units, so there are no mutual connections in the layers. In tasks with images the visible units corresponds to pixels, one visible unit has one pixel. The hidden units are conditionally independent given the visible states, this makes each hidden unit an independent feature.

The training of the RBM is performed by using the contrastive divergence(CD) procedure. This procedure consists of three steps:

1. Clamping the visible units with an input vector and update hidden units
2. Updating all the visible units in parallel to getting a reconstruction
3. Collecting the statistics for correlation after some steps using mini-batches and updating weights

The RBM for binary-type MNIST images was trained as described above, by using contrastive divergence learning and the reconstruction performance of the RBM was analyzed. Figure 1 below show how well the network reconstructed the images for each digit in the training set.

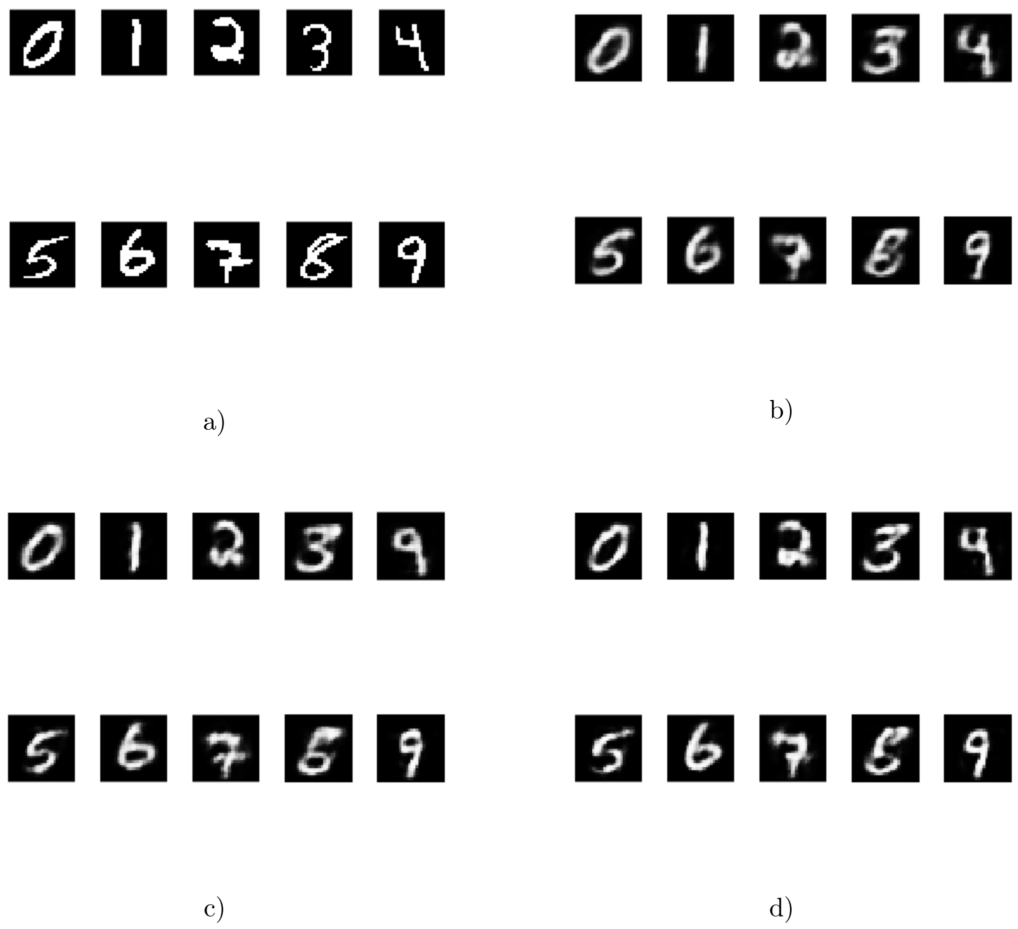

Figure 1: Figure a) represents training samples reshaped to images, figure b) represents the reconstructed images with 50 units, c) represents the reconstructed images with 75 units and d) with 100 units, all for 100 epochs

Some interesting things show up in figure 1c. The digits "4" and "8" look like the digits "9" and "5" respectively. This is due to the digits sharing many common features in the original dataset. Because these digit strongly resemble each other in the dataset they are very likely to have similar distributions over the RBM and therefore end up reconstructing an incorrect digit. The other have more distinct features and are thus less likely to get mixed up.

Furthermore, judging from figure 1, increasing the number of nodes seems to improve the accuracy of the reconstruction. Figure 2 below shows a quantitative measure of the error as an RMSE. As can be seen, the error indeed decreases as the number of nodes decreases.

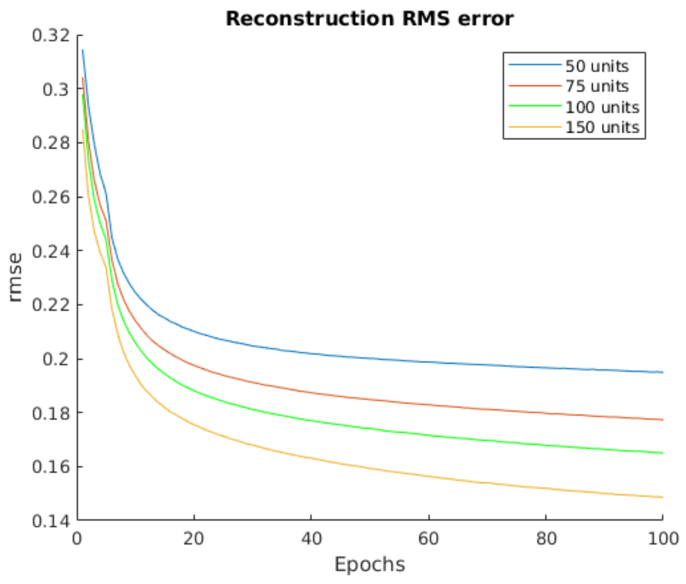

Figure 2: The root mean square error for the reconstruction for different number of nodes

The test set was reconstructed to check whether or not the network was overfitting the data. Table 1 shows the final reconstruction error for the test data for the trained network.

Table 1: Test set reconstruction error

Units	50	75	100	150
Reconstruction error	0.1978	0.1818	0.1703	0.1556

As seen, the reconstruction error for both the training and test data sets are very close to each other, meaning that the network with high certainty has not overfitted the data.

Autoencoder

Autoencoders are networks which encodes the given data, to then be able to reconstruct the same data by decoding. This property can be used to compress a large amount of data to a more compact form, similiar to PCA. Here we tried using an autoencoder with one layer trained with scaled conjugate backprop using the training data to see how well it reconstructed the images.

The network was also tested using the test set to check how well it performed on a set that the network had not yet seen. Figure 3 below shows the result.

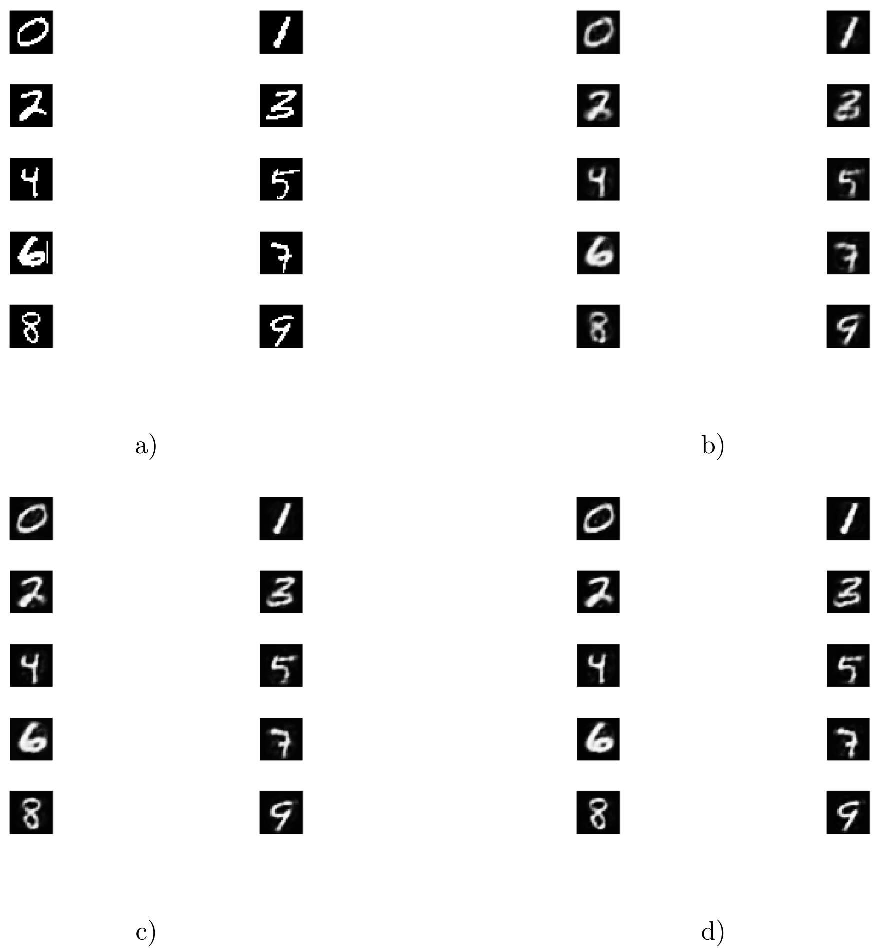

Figure 3: The original image and reconstructed images. a) is the original testing images, b), c) and d) show the reconstructed images for 50, 100 and 150 hidden units respectively.

This time, there were no obvious mix-ups in digits. One reason might be that the autoencoder was trained for 150 epochs while the RBM was trained for 100. The error was once again quantified, this time by MSE. The error is shown in figure 4.

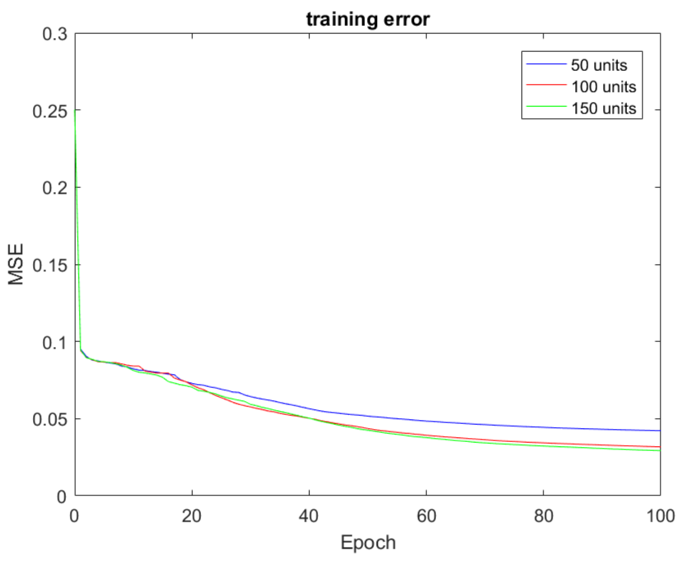

Figure 4: The mse while training for 50 hidden units, 100 hidden units and 150 hidden units

As in the case with RBM, it can be seen that increasing the number of nodes decreases the error.

An autoencoder (and also an RBM) works by responding to features in the layer above it. This information is stored in the weights. It is therefore of interest to visualize each neuron's weights to see how the features are stored. This was done for the network with 100 hidden neurons. The result is shown in figure 5 below.

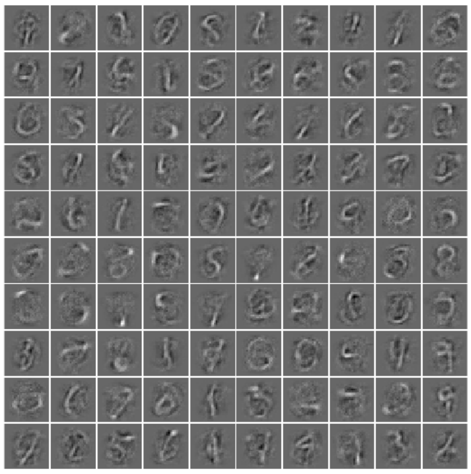

Figure 5: The autoencoder weights with 100 hidden units

As can be seen, the weights store combinations of stroke patterns from the images and a couple of the resemble whole digits.

Classification

Deep network architectures can be used for classification. Here we extended the network architectures used for reconstruction by adding more hidden layers as well as an extra extra output layer at the top for handling the classification. The output layer used here was a softmax logistic regression in both cases. The use of softmax was found appropriate because a softmax is suitable for classifying an input into a number of different classes.

The training was done in a layer-by-layer fashion. The training data is first clamped onto the visible units to perform unsupervised training on the first hidden layer. When this is done, the nodes of trained hidden layer are used as visible nodes for the next hidden layer an so on. When all the hidden layers have been trained, the last hidden layer is used as an input to train the final classification layer. The last training is done with a target, i.e. it is supervised. The softmax layer was trained using cross entropy.

To measure the accuracy of the net we simply checked the percentage of correct classifications made by the network.

DBN

A Deep Belief Network is a a multi-layer network where each layer is a RBM stacked on top of each other. The first step of the training for a DBN is to clamp data into the visual nodes and learn the features of the data by using the contrastive divergence procedure. When the output of the hidden layer is obtained it can be used as a visible layer's input to the next hidden layer and so on depending on how many layers of RBM you have. A DBN alone can't be used for multi class classification purposes and therefore, as mentioned above, a new network of supervised learning has to be added on top of the last hidden layer, which in this case was a softmax layer.

The DBN was trained with different amount of RBM layers and units. The results are depicted on the table below.

Table 2: Accuracy of the network for different architectures

Num. of RBM	Architecture	Training Accuracy	Test Accuracy
1	100→softmax	94.50%	89.80%
2	100→ 100 → softmax	95.04%	91.30%
2	100→ 50 → softmax	91.33 %	91.60 %
3	100→ 100 → 100 → softmax	95.24 %	90.80 %
3	100→ 100 → 50 → softmax	92.65 %	90.65 %
3	100→ 50 → 50 → softmax	91.33 %	89.80 %

Stacked Autoencoders

A deep network structure can also be constructed by stacking autoencoders on top of each other. A layer closer to the bottom encodes the input from the level below it and passes this on to the layer above it. The last encoder passes its output on to a softmax layer which classifies the input. The size of the first layer was chosen to be 100. The classification was done once again using 1,2 and 3 layers. The results are shown in the table 2 below.

Table 3: Accuracy of the network for different architectures

Num. of AE	Architecture	Training Accuracy	Test Accuracy
1	100→softmax	92.50%	88.73%
2	100→ 100 → softmax	94.38%	89.55%
2	100→ 50 → softmax	93.04 %	90.12 %
3	100→ 100 → 100 → softmax	93.35 %	90.60 %
3	100→ 100 → 50 → softmax	90.75 %	89.45 %
3	100→ 50 → 50 → softmax	90.65 %	88.55 %

Discussion and Conclusion

The reconstruction error decreased for the RBM and the autoencoder when we added more nodes into the network, which can be seen in the plots figure 2 and figure 4. And comparing with the reconstructed images figure 1 and figure 3, we can clearly see that the digits became sharper and more recognizable as we increased the network's complexity. Eg. in figure 1c, the image that should represent the number 4, instead looks like a 9. But in figure 1d, the image clearly resembles the correct number.

The trained RBM reconstructed the images pretty well, but we still got a pretty high reconstruction error, see figure 2. Compared with the error that the encoder produced see figure 4, which seems more accurate as we are dealing with networks that can handle complex data such as MNIST. One thing we could have done was to use grid search to obtain the optimal parameters for our RBM. However, we tested the network with the given test set. Table 1 shows the reconstruction error for the test-set. It is possible to see that 150 units gave the smallest reconstruction error for both training and testing. Also, comparing the errors shown in figure 2 for the training data and table 1 for the test data, they are very close to each other. This indicates that the possibility of overfit is low.

Then we added more layers to the networks, DBN and stacked autoencoder, for more expressive power and compactness. The results of the performance are shown in tables 2 and 3. With the DBN, the architecture that showed best training accuracy was the one with the highest complexity, 3 layers with 100 nodes each. Following up is the one with 2 layers containing 100 nodes each. Similar patterns can be seen with the stacked autoencoders. Thus, adding more layers does not necessarily increase the training accuracy, instead it is the amount of neurons used in each layer individually. Probably due to the concern of overfit. And regarding the test accuracy, using 2 layers of RBM produced the best result and for the stacked autoencoder with 3 layers containing 100 nodes each.

Last we compared the performance of the DBN and the stacked auto encoder with 2 layers, with an MLP architecture consisting also with 2 hidden layers.

Table 4: Accuracy of MLP with different nodes

Architecture	Training Accuracy	Test Accuracy
50→50	79.44%	77.40%
75→ 75	83.26%	82.35%
100→ 100	84.04 %	81.30 %
150→ 150	86.01 %	82.15 %

Overall, the performance of the MLP was worse than the DBN and the stacked auto encoder. A more complex MLP could show similar performance, but would require more training time comparing with the DNNs.

Although the accuracies were good, we think it can still be improved. This can be done by experimenting with the number of layers, the size of the layers, using different training methods, different error measures and different classifiers.