Artificial Neural Networks & Deep Learning

Master of Artificial Intelligence

Exercise Sessions Report

Author: Guillaume Lamine (r0737163)

June 3, 2019

1 Exercise Session 1 : Supervised learning and generalization

1.1 Exercise 1

1.1.1 Function approximation

An Artificial Neural Network (ANN) is trained with the help of the MATLAB library for neural networks.

The goal is to approximate the function $y = \sin(x^2)$ for $x = 0 : 0.05 : 3\pi$. Different optimization algorithms will be compared to the classical gradient descent (gd). For all the training algorithm, 3 simulations are run : for 1, 15 and 1000 epochs.

Overall, the gradient descent algorithm has clearly the slowest convergence rate. On the opposite, the Levenberg-Marquart (lm) algorithm performs the best and has the best correlation with the initial data in this setup. The comparison between best and worse can be seen in Figure 1.

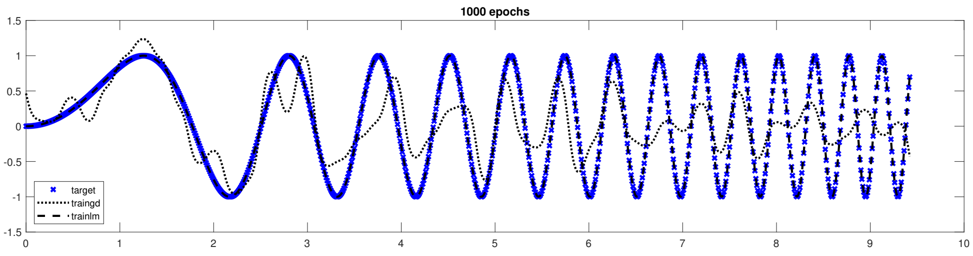


Figure 1

As the data is not noisy at all, there is no question of overfitting. The algorithm can be run for many epochs. On the Figure, we can see that after 1000 epochs, the lm algorithm is perfectly matching the data. This is barely the case for the gd.

1.1.2 Learning with noisy data : generalization

Now, some noise is added to the data. Still, the algorithm to perform the best is the Levenberg-Marquart, and the worst is the gradient descent. Results of the approximation after 1000 epochs is shown on Figure 2.

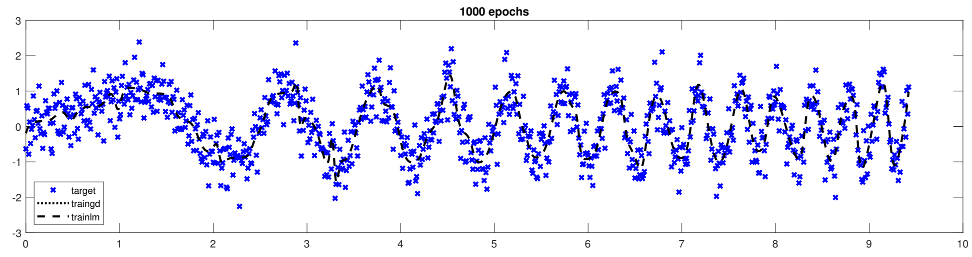


Figure 2

1.2 Exercise 2 : Personal regression example

In the second exercise, the dataset at hand comes from a non-linear function with two input dimensions and one output dimension. We have 13600 data points. From it, we sample a training set, a validation set and a test set, all of 1000 points.

First, the data of the training set can be visualized to gain some insight on the data. It will also allow us to compare to our final approximated surface. The training set is shown on Figure 3.

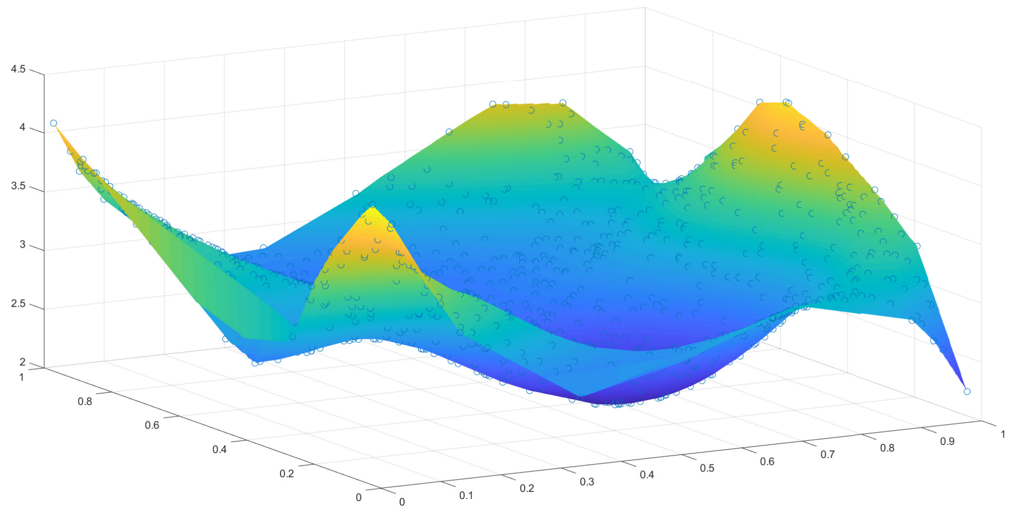


Figure 3

A neural network architecture can now be build and trained by using the training and validation sets. For The investigation of a good architecture, there are only 3 parameters that are going to be played on : the number of layers, the number of hidden units per layer and the training algorithm. Note that we only focus on fully connected neural network in this exercise session.

For the training algorithm, there are only 2 that will be compared here : gradient descent, the classic one and Levenberg-Marquart, as we say that it is most likely the one that will give the best accuracy. We only study 3 architecture and in each, both training algorithm are tested.

For the choice of architecture, the analysis focus here on the question : for the same amount of neurons, in this current case, is it better to have a deep network with small layers or a shallow one with larger layers ? For doing that, 3 architectures are tested, all composed of 50 neurons in total. The first is one layer of 50 neurons ([50]), the second is two layers of 25 neurons ([25 25]) and the third, five layers of 10 neurons ([10 10 10 10 10]).

The results of the analysis are shown on Figures 4, 5a and 5b.

1.2.1 Analysis of the results

In every case, the Levenberg-Marquart algorithm is performing better after 1000 epochs. The architecture giving the best results on the validation set is the deep architecture of 5 layers of 10 neurons. Moreover, the algorithm is not even yet overfitting after 1000 epochs because the validation error has not yet reached a minimum. This is not the case for the two other architectures where lm error on validation reaches a minimum. Concerning the gradient descent, in every case, it has not yet reached a minimum after 1000 epochs.

For these reasons, the best combination found is the deep network trained with Levenberg-Marqart.

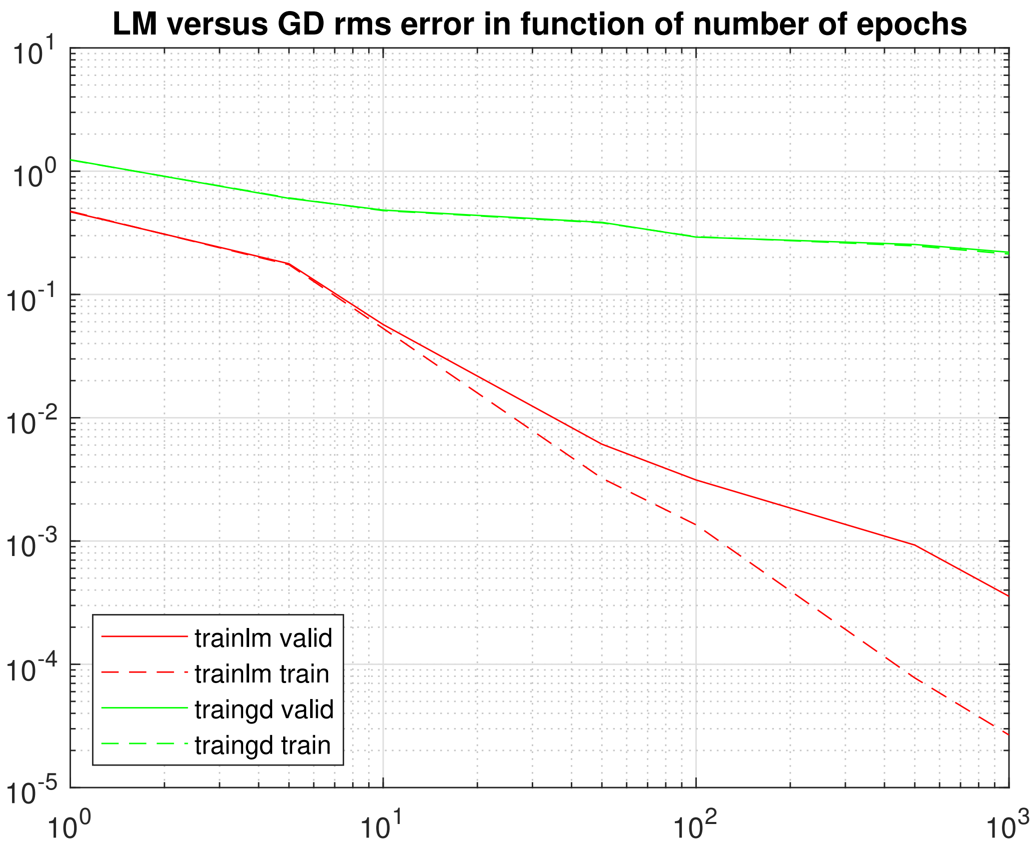
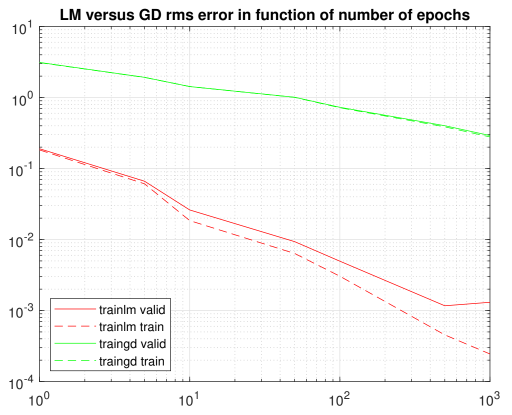
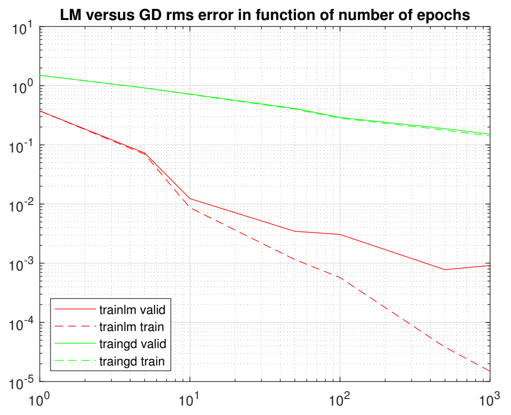


Figure 4



(a)

(b)

Figure 5

Finally, the predictions are made on the test set to visualize the final results. The result is displayed on Figure 6. Note that the colored surface is the approximation surface and and that the small circles are the data points of the test set.

Visually, the results appear to be very close to the tested data.

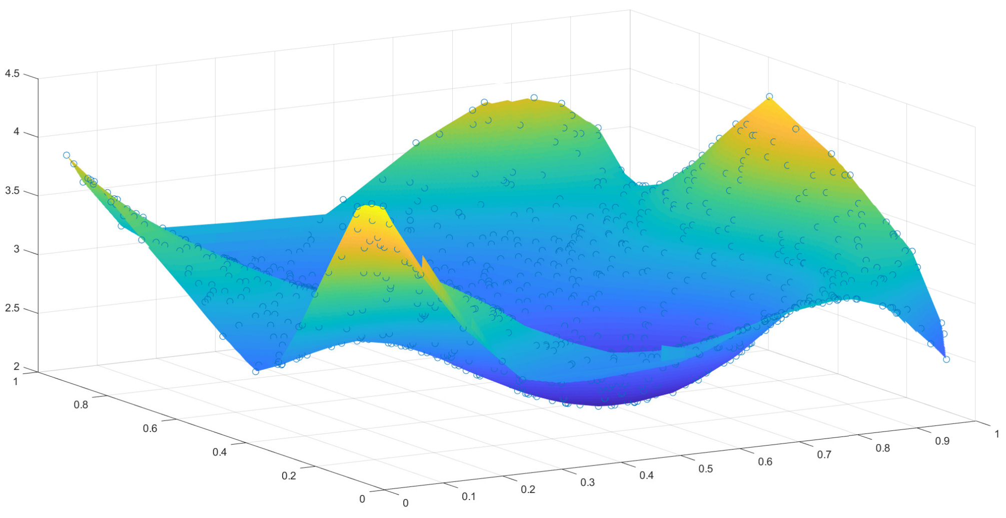


Figure 6

2 Exercise Session 2

2.1 Exercise 1 : Hopfield networks for handwritten digits

The Hopfield network used for the handwritten digits dataset is really robust to noise. However, it was still possible to make it mismatch two noisy images. The result obtained on Figure 7 was generated with a noise level of 5. (It is highly out of the range $[0, 1]$)

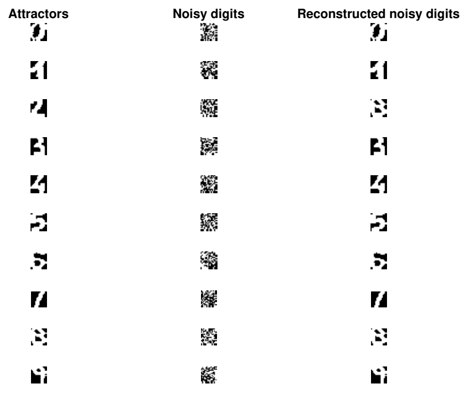


Figure 7

2.2 Exercise 2 : Long Short-Term Memory networks for time series predictions

In this exercise, two types of models are tested to do prediction on time series. The dataset studied is the Santa Fe dataset. The goal is to train a model on a 1000 data points and try and predict the 100 last new points.

2.2.1 Use of a fully connected Neural Network

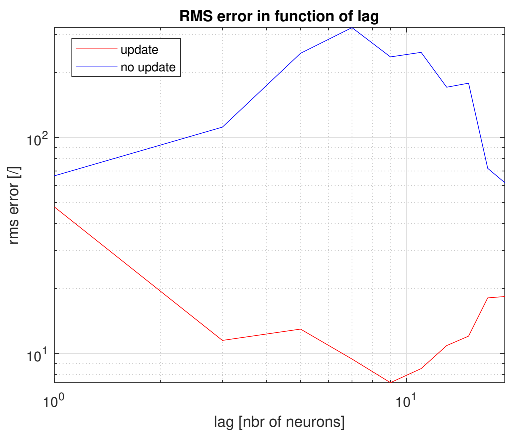
The first model is a fully connected feed-forward network with one hidden layer. In this case, the prediction for a certain time t is equal to a weighted sum of the previous values up to a certain lag p. Here p is the size of the input layer of the network. Thanks to the function `getTimeSeriesTrainData`, the data is nicely adapted for the training of the NN.

There can be two versions for the model. The predictions of the 100 points can be done with or without updating the values on the previously predicted points. This variation is of course of great importance in the precision of the 100 points test set.

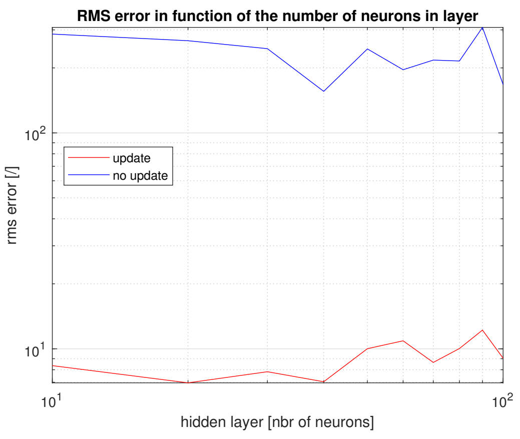
Before predicting with the final results, the model performances are investigated through two parameters : the lag and the number of neurons of the model (knowing that only a one layer network is considered).

The comparison for multiple values is shown on Figures 8a and 8b.

The results for the NN are shown on Figures 9a and



(a)

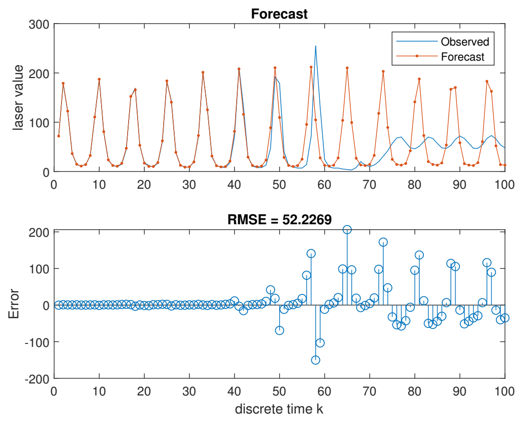


(b)

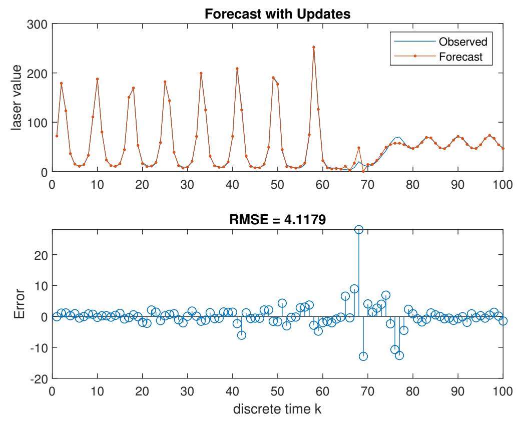
Figure 8

The best performances measured based on the RMS error are found for the cases with and without updates. With updates, the best combination is : $p = 9$ and $n = 40$ and without updates the best is $p = 20$ and $n = 40$.

Now, the predictions are made on the test set with and without updates. (See Figures 9a and 9b for results)



(a)



(b)

Figure 9

2.2.2 LSTM network

The second model used to do time predictions is the Long Short-Term Memory network (LSTM). It is especially good at learning long-term dependencies. The only parameter that is investigated in this exercise is the number of hidden units in the model.

The results are shown in Figure 10.

In the chose range of study for the number of hidden units, the best values are 250 and 300 for model with and without updates respectively.

The predictions on the test set with the LSTM model are shown on Figures 11a and 11b.

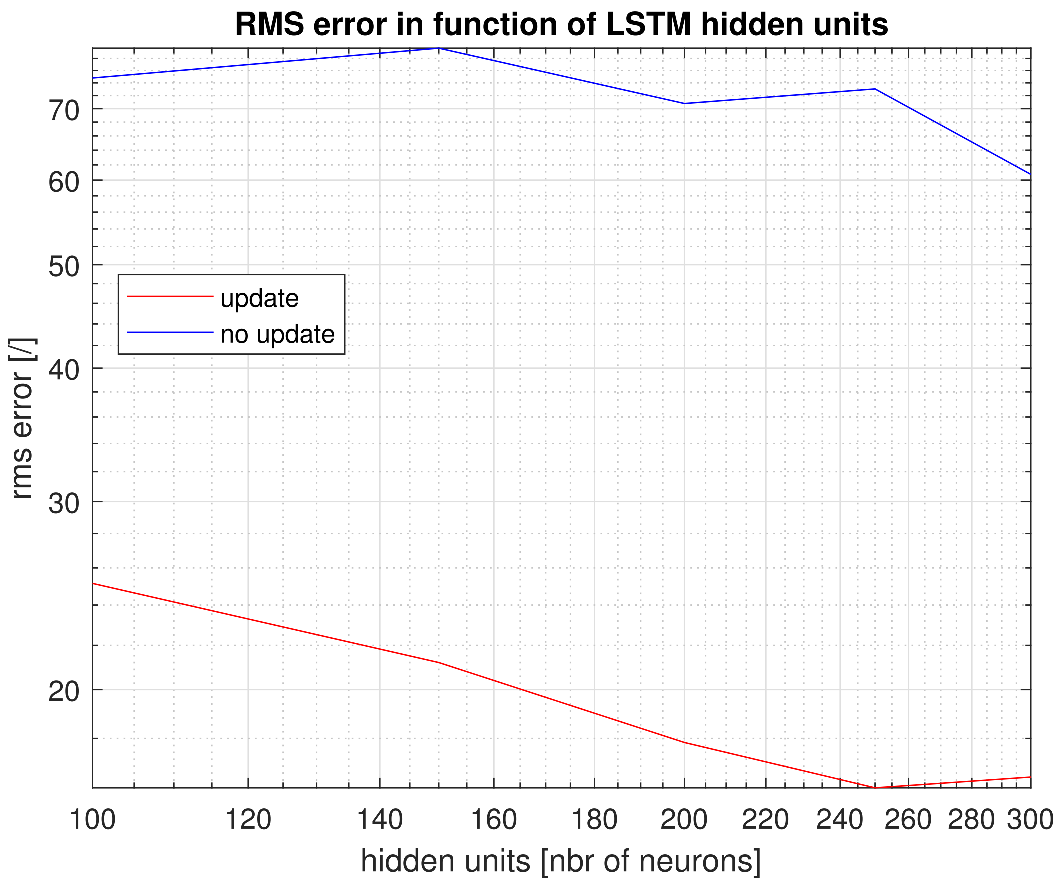
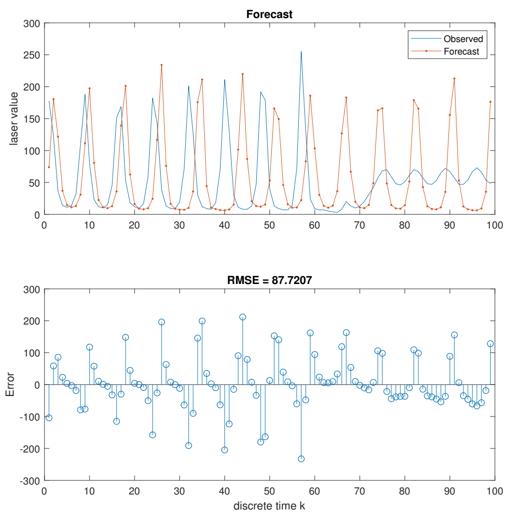
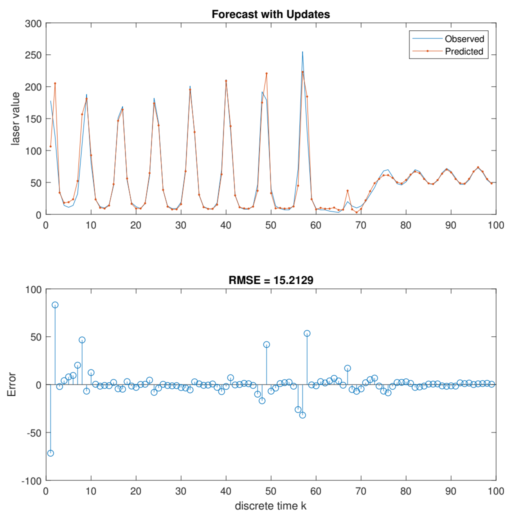


Figure 10: LSTM rms error in function of the number of hidden units in the model



(a)



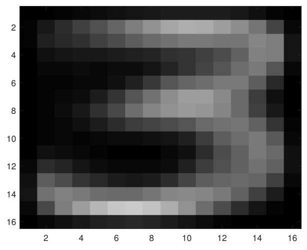
(b)

Figure 11

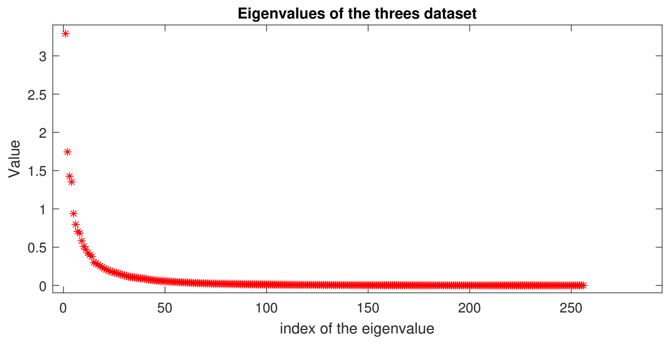
3 Exercise Session 3

3.1 Principal Component Analysis on handwritten data

In this exercise, a Principal Component Analysis is performed on a handwritten digits dataset, in order to extract the main features out of the pixels of images. To gain some better insight on the dataset for a PCA, the mean and the distribution of the eigenvalues is plotted on Figures 12a and 12b.



(a)

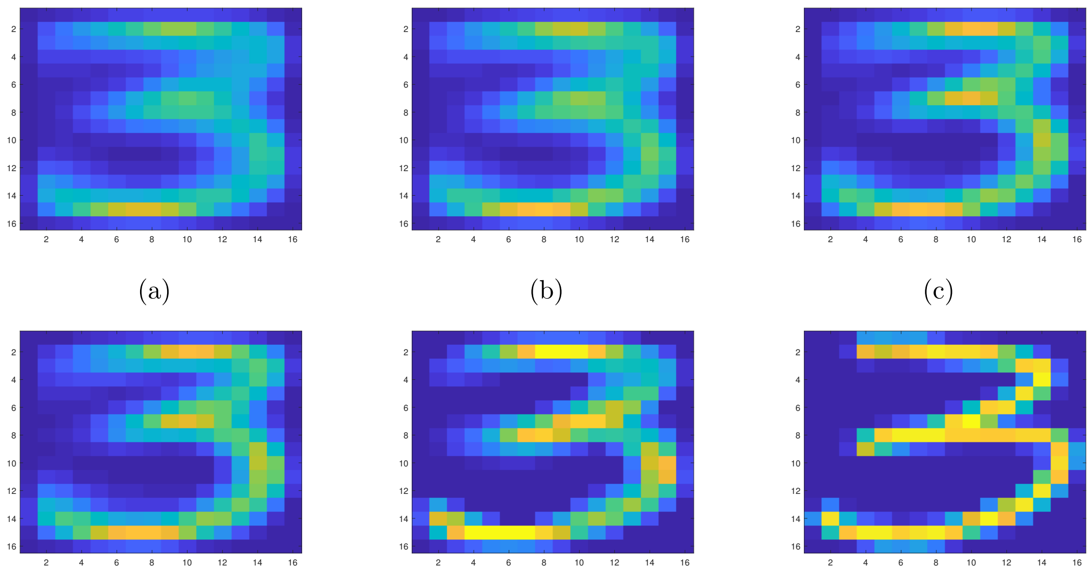


(b)

Figure 12

Two first observations are that already on the mean, the 3 is very distinguishable, which means that the dataset is very clean. This will probably allow the PCA to give good results with few eigenvalues. Moreover, the high values eigenvalues are mainly among the first 10 ones. Ten eigenvalues will most probably be enough to have small errors on reconstruction.

The next step is to test the PCA algorithm and reconstruct images after compressing them to the four first eigenvalues. The reconstructed images are shown on Figures 13a to 13d. Figures 13e and 13f show respectively the reconstruction with 10 eigenvalues and the real image to reconstruct.



(a) (b) (c)

(d) (e) (f)

Figure 13

Finally, a last interesting result is to plot the RMS error curve depending on the number of eigenvalues used and compare it to the cumulative sum of the eigenvalues. On Figure 14, we can observe that the error logically goes to zero when k reaches 256 as it is the total number of pixels of the image.

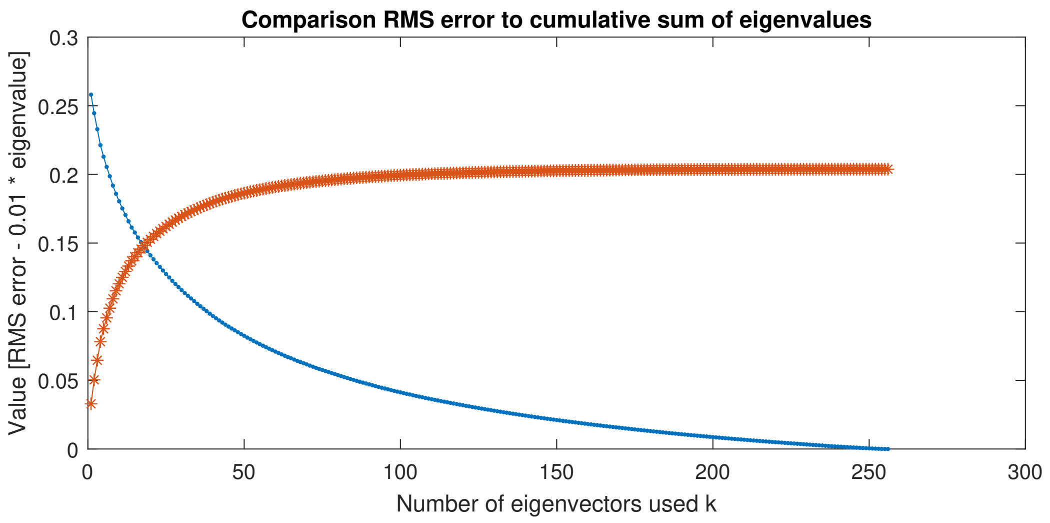


Figure 14

3.2 Stacked Autoencoders

In this exercise, we use autoencoder architectures to learn handwritten digit classification. In our case, the analysis will limit to three particular architectures of Stacked Autoencoders (SAE). We will denote them by their layers and number of units per layer. Here the three architectures are :

- Archi 1 : input layer = 784; layer1 = 100; layer2 = 50; layer3 = 10
- Archi 2 : input layer = 784; layer1 = 400; layer2 = 100; layer3 = 50; layer4 = 10
- Archi 3 : input layer = 784; layer1 = 100; layer2 = 10

The first architecture was the one given in `DigitClassification.m`. We assume that the parameters $MaxEpochs$, number of units, .. are well chosen, so we keep them as such. For the bigger architecture 2, we will use a rule of thumb to more or less augment proportionally the parameters with respect to the size of the new stack of autoencoder. However, the number of epochs of such a large stack is limited to 300 to avoid too long training time. As bigger networks need more time to train, results could probably be improved by augmenting the number of epochs.

For the smaller architecture 3, the parameters are kept the same, taken from architecture 1.

The final results are given in Table 1. In each case, the SAE is compared to its equivalent (in terms of number of layers and neurons) of fully connected feed-forward network, trained all layers at once.

	fine tuning	no fine tuning	Pattern Net equivalent
SAE 1	99.7 %	82.1 %	96.6 %
SAE 2	99.8 %	81.3 %	96.2 %
SAE 3	99 %	98 %	97 %

Table 1

It can be observed that the deeper SAE is giving better results than the initial one. On the contrary, the shallower has worse results. In every case, they are performing better than the PN equivalents. Note still that the results for the PN 3 are more variable and subject to the initialization weights. Finally, fine tuning is clearly a key to reach good results for a SAE, and it is even more the case when more stacks are added.

3.3 Convolutional Neural Networks

3.3.1 Answer to theoretical questions

First, the weights of the convolutional layers represent the weights of the filters that are learned and applied convolutionally on the images.

Secondly, the downloaded network is the AlexNet. This network transform the input from 227 x 227 x 3 to 6 x 6 x 256 just before entering the layer 6, the first layer of fully connected networks.

Thirdly, the final output dimension of AlexNet is 1000. This is more or less 154 times less than the input : $227 \cdot 227 \cdot 3 = 154587$.

3.3.2 CNN for handwritten digits

In this section, a few different CNN architectures are investigated in order to compare their accuracies. There only three architecture that are being tested here :

- Archi 1 : imageInputLayer([28 28 1]); convolution2dLayer(5,20); reluLayer; maxPooling2dLayer(2,'Stride',2); fullyConnectedLayer(10); softmaxLayer; classificationLayer().
- Archi 2 : imageInputLayer([28 28 1]); convolution2dLayer(5,12); reluLayer; maxPooling2dLayer(2,'Stride',2); convolution2dLayer(5,24); reluLayer; fullyConnectedLayer(10); softmaxLayer; classificationLayer();
- Archi 3 : imageInputLayer([28 28 1]); convolution2dLayer(5,12); reluLayer; maxPooling2dLayer(2,'Stride',2); convolution2dLayer(5,24); reluLayer; maxPooling2dLayer(2,'Stride',2); convolution2dLayer(3,48); reluLayer; fullyConnectedLayer(10); softmaxLayer; classificationLayer()

In summary, these are architectures with 1, 2 and 3 layers of convolutions. Archi 2 was the initial one. For Archi 3, an additional convolutional layer was added at the end of the basic Archi 2. Archi 1 is a simple case of a smaller architecture.

The small Table 2 gathers the different precisions for the 3 architectures, depending on the number of training epochs.

	15 epochs	30 epochs	60 epochs	120 epochs
CNN 1	98.12 %	98.72 %	99 %	99.12 %
CNN 2	98.8 %	99.4 %	99.6 %	99.64 %
CNN 3	24.36 %	69.44 %	98.52 %	99.68 %

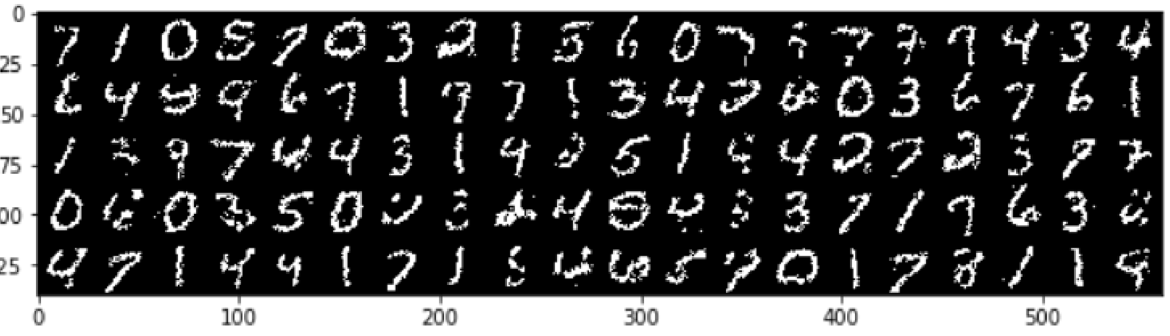
Table 2

The results show that the largest architecture can reach the best performance but only after 120 epochs of training. The smaller architecture converges quickly to high performance but stagnate further because of the lack of complexity of the learner.

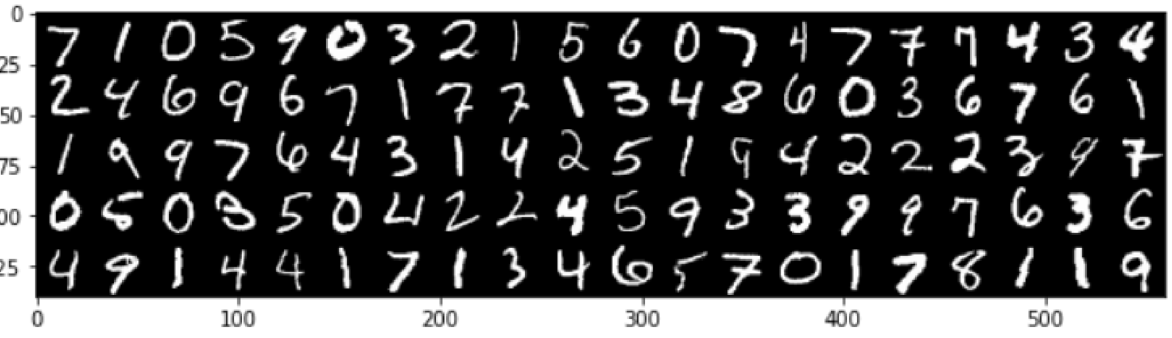
4 Exercise Session 4 : Generative models

4.1 Restricted Boltzmann Machines

The best results for RBM were obtained for the following combination of parameters : #components = 200, learning rate = 0.01, iterations = 15. Results are shown on Figures 15a and 15b.



(a)

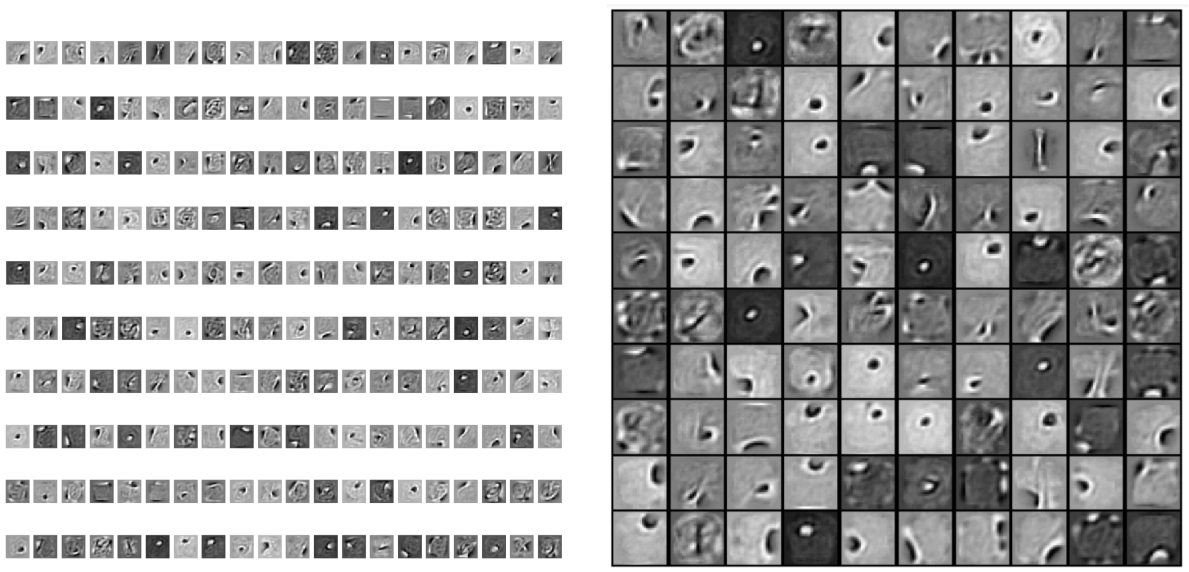


(b)

Figure 15

4.2 Deep Boltzmann Machines

The Deep Boltzmann machine here consists of two RBM stacked on top of each other. On Figures 16a and 16b it is possible to visually compare the weights of the first layer of the DBM to the layer of the RBM.



(a) (b)

Figure 16

Finally, we can generate new handwritten numbers by sampling the learned distribution. The results is clearly better than the simple RBM. Results are shown on Figure 17.

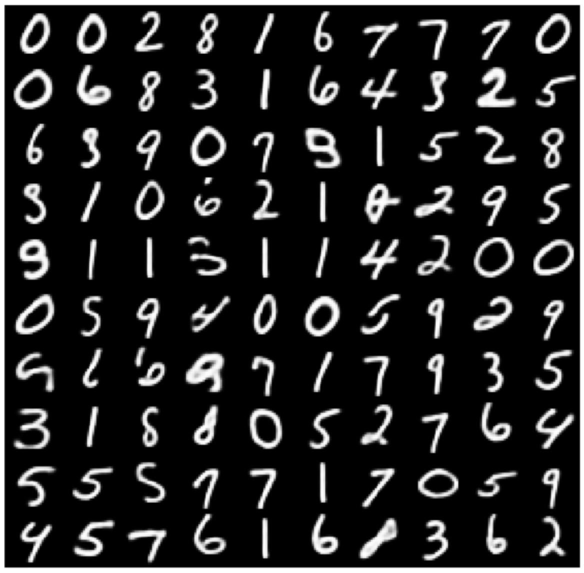


Figure 17

4.3 DCGAN's

In this section we show the kind of results that are given by a Generative Adversarial Network. The results shown on Figure 18 are obtained by training during 20000 batches.

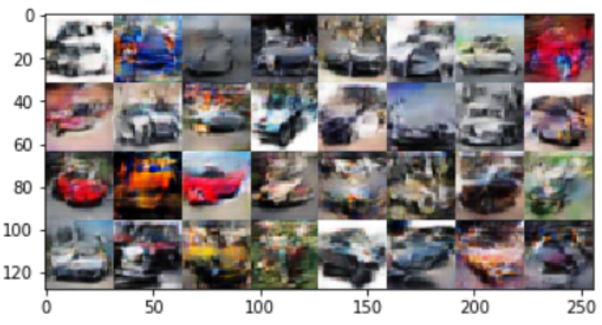


Figure 18