Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 4
Deep neural network architectures with autoencoders

Samuel Leonardo Gracio and Martin Verstraete

October 12, 2018

1 Main objectives and scope of the assignment

List here a concise list of your major intended goals, what you planned to do and what you wanted to learn/what problems you were set to address or investigate, e.g.
Our major goals in the assignment were

- to familiarize with the Hopfield network and understand how it was possible to recall detailed images in a few iterations
- to understand the concept of energy

Methods

We worked with Python and with a Google Colab environment. We only used numpy functions and matplotlib.

1.1 Autoencoder for binary-type MNIST images

We did train an autoencoder with just a single hidden layer. For that, we used the stochastic gradient descent with a constant learning rate and regularization, as it was recommended in this lab.

To choose the number of hidden nodes, we wanted an number of hidden nodes lower than 784, the dimension of the input space.

There are some of the results we had. It's a graphic representation of the error as the sum of the mean error between the original input and the reconstructed input after a training through 100 epochs.

Let's now see the influence of the number of hidden nodes on the learning and testing errors:

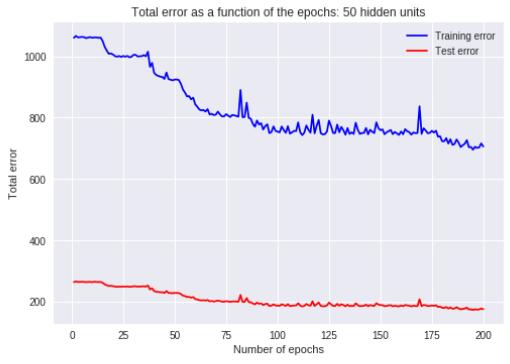
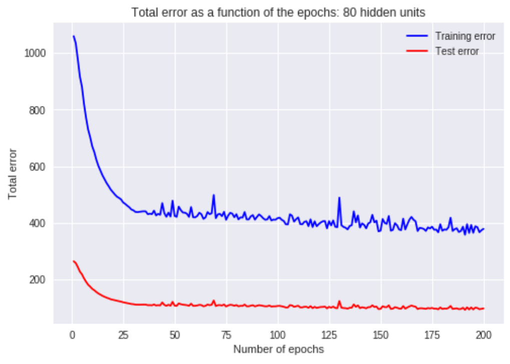


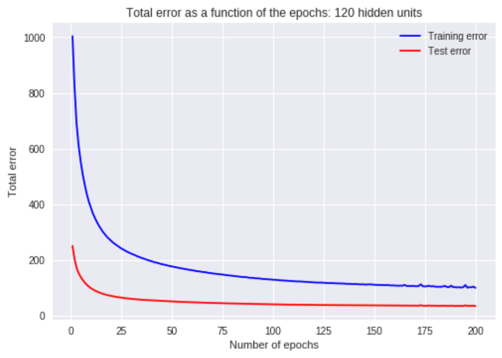
Figure 1 : Total error as a function of epochs for 50 hidden units

Here, we did it with 50 hidden units. As we can see, the total training error converges to 1059 and the total testing error to 264… First of all, it can be weird to find more errors in the training set than in the test set but this is due to the number of data in each of these datasets.These are not good results. We should increase the number of hidden units.



This is the result with 80 hidden units. The total training error converges to 706 and the total testing error to 176. It's better than before but we can still improve this.

Figure 2 : Total error as a function of epochs for 80 hidden units



Here, we have 120 hidden units. We have a really good looking curve. The total training error converges to 379 and the total testing error to 98, which is good. We have almost four time more errors in the training set so we're not facing overfitting.

Figure 3 : Total error as a function of epochs for 120 hidden units

After computing the reconstructed images, we can display both original and reconstructed images, for each digit :

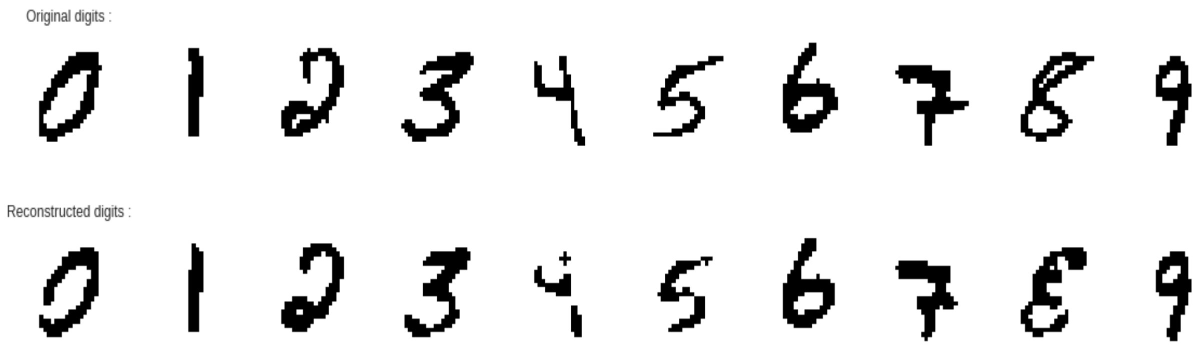


Figure 4 : Original and reconstructed images

They look good, we can still recognize each number (maybe it's more difficult for the 4 and the 8). That means that our network is working well.

Thus, you need enough hidden nodes to be able to reconstruct correctly the input images.

Now we will think about sparsity. Measuring sparseness is basically knowing how many neurons are really active in your hidden layer. We compute the activity of a neuron by comparing it to a certain value. Under this certain value, it's inactive. Over it, it's active. Here, it's active if it's equal to 1, unactive if less than that. Here, we have a graphical representation of that :

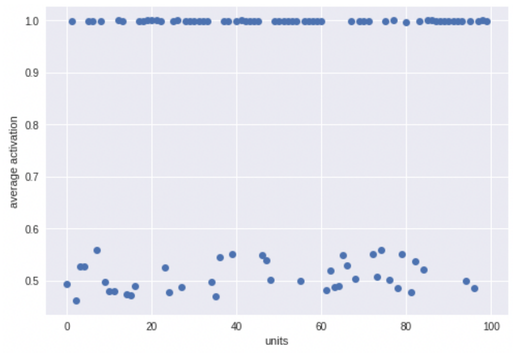


Figure 4 : Sparseness for a hidden layer with 120 hidden nodes.

What we can see here is that you have many neurons inactive, like 30% of them. It's a good compromise between a reasonable number of nodes and a good level of sparsity.

We can also look into sparsity by plotting the weight vectors :

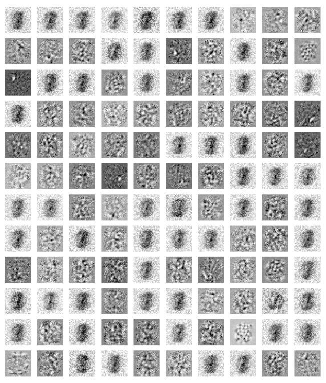
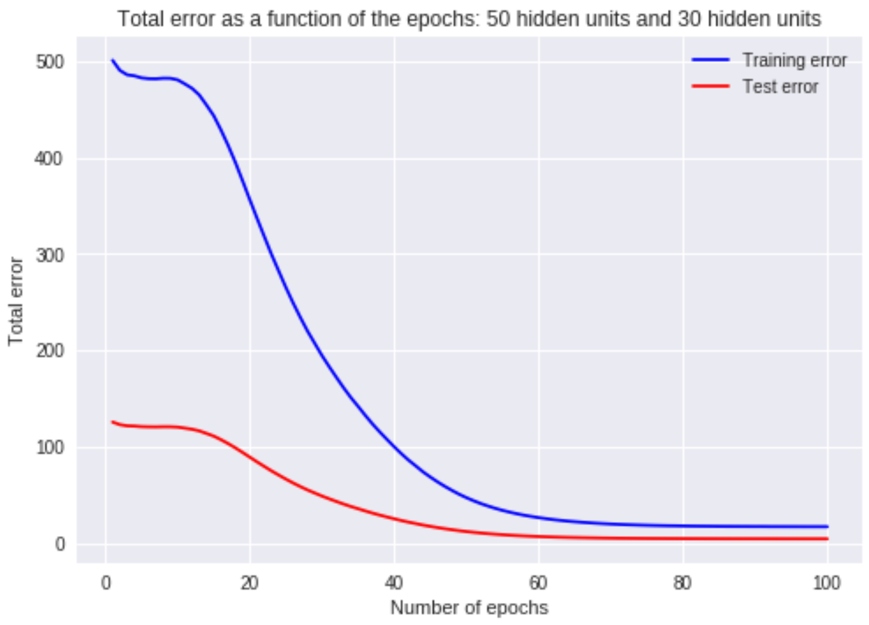


Figure 5 : Weight vectors for 120 hidden nodes

Here we can see two kind of patterns : the one with a big black center and the others. The first group gives no real information since they are the same, they all code the same information.

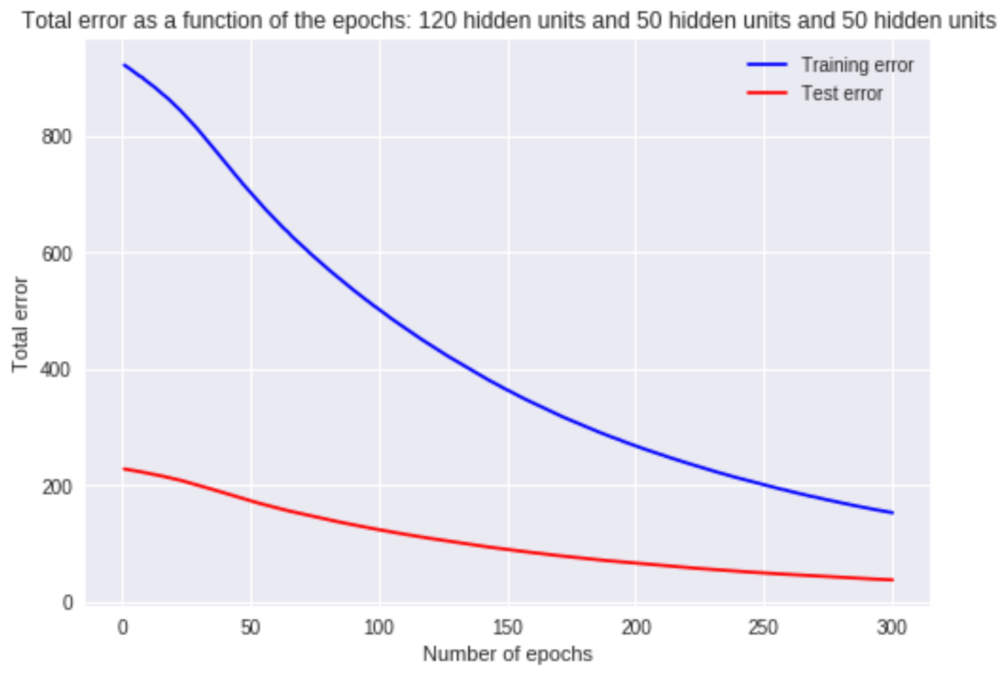
3.2 Stacked autoencoders for MNIST digit classification

We try different values of hidden nodes for each layer of autoencoder and we plot the learning and error curve to see which combination is effective.



The combination 50 hidden nodes on the first autoencoder and 30 on the second gives some good results.

We can try the same with three layers:



The combination 120-50-50 is the one that gives the best results.
If we use this network to predict the outputs of the testing data, we find a score of 0.86.

```
clf.score(hidden_3_act_dict_tst['120_50_50'], targetdigit_tst)
```

```
0.86
```

We can know fine-tune the network with a backpropagation in order to improve the prediction. Doing so, we know get a score of 0.93 which is much better.

```
ae.score(bindigit_tst, targetdigit_tst)
```

```
0.934
```

Thus, the three layers of autoencoders with greedy layer-wise pre-training enable us to improve the performance of the classifier by avoiding gradient vanishing.

4 Remarks

It was really interesting to understand what were inside the hidden layers. The only problem of this lab was the computing time.