Report of homework 4: autoencoders with pytorch

Clara Eminente

February 1, 2021

1. Introduction

In this report the basic structure of an autoencoder provided during Lab05 is expanded and the features of this unsupervised learning method are studied and tested over different versions of the *MNIST* dataset.
More specifically, in the first part of this report the architecture of the autoencoder is explained and, using a random search procedure, the best hyperparameters for the model are selected. Moreover, the role of the dimension of the encoded space is investigated by training the network for different values of this parameter.
In the second part, after fixing the model, its performances over corrupted data are explored and a denoising version of the architecture is implemented.
In the final part, the properties of the encoded space are studied, providing some insight on the properties of this type of network.
The most part of this homework was developed and run on Google Colaboratory in order to be able to use GPU acceleration via *Pytorch* and *Cuda*.

2. Architecture

As mentioned before, the basic architecture of the autoencoder corresponds to that provided during the fifth laboratory session. The main adjustments to the main class *Autoencoder* were the introduction of dropout and some functions which will be described during this report. Among these, a training functions that wraps up the already implemented functions for training and testing was added. This wrap up trains the network for one epoch and includes the possibility to early stop the training monitoring the performances using the test (validation) set. Early stopping was implemented from scratch (see Homeworks 1 and 2).
For the rest of this report the loss function of the model is the *MSE* (reconstruction error) and the optimizer used for training is *ADAM*. The hyperparametrs to be searched are the learning rate (*lr*), the *L2* weight decay and the dropout fraction. Moreover, as mentioned before, the role of the encoded space dimension will be investigated.

The dataset used to train and test the model is the *MNIST*. At first two samples are loaded as train (60000 elements) and test (10000) sets. The test set will be used just at the end to test the performance of the final model and on corrupted date (see next sections).

2.1. Random search

The first task is finding a good set of hyperparameters (learning rate, L2 weight decay and dropout fraction) using a random search procedure. The dimension of the encoded space is however expected to play the most important role in the performance of the model. It was thus decided to set the dimension of the encoded space to 4, perform a random search over only 30 different models and set for the best set among these, even though the search is not the most exhaustive. This was also considered the best choice considering that training was computationally very intensive and runtimes ended up being very long.
The network was trained for 45 epochs using cross validation over three folds of the training set. The model with the least average validation loss over the folds was selected as the best one.
Table 1 shows the range in which the parameters were sampled and their best value.

parameter	**range**	**best**
learning rate	$5 \cdot 10^{-4} - 5 \cdot 10^{-2}$	$1.2 \cdot 10^{-3}$
L2 weight	$5 \cdot 10^{-5} - 5 \cdot 10^{-3}$	$8.8 \cdot 10^{-5}$
dropout fraction	0-0.5	0.15

Table 1: *Searched parameters*

2.2. Encoded space dimension

An autoencoder is said to be *overcomplete* if the dimension of its encoded space (*enc_dim*) is greater of equal to that of the input. In this case the autoencoder has enough free parameters to just learn

to copy the input. An *undercomplete* autoencoder, however, has to learn to extract the basic features of the dataset. The higher the encoded space dimension the more "precise" the encoded representation: this is why the dimension of the encoded space is expected to play a key role in the performances of the model.
In order to explore this feature the best set of hyperparameters previously found is used to train the network with different values of *enc_dim*. No cross validation is used but a validation set is kept for early stopping in order to avoid overfitting. The network is trained for 80 epochs for each value of *enc_dim*. Figure 1 shows the final loss of the model for different values of *enc_dim*. As expected, it decreases sensibly as *enc_dim* grows.

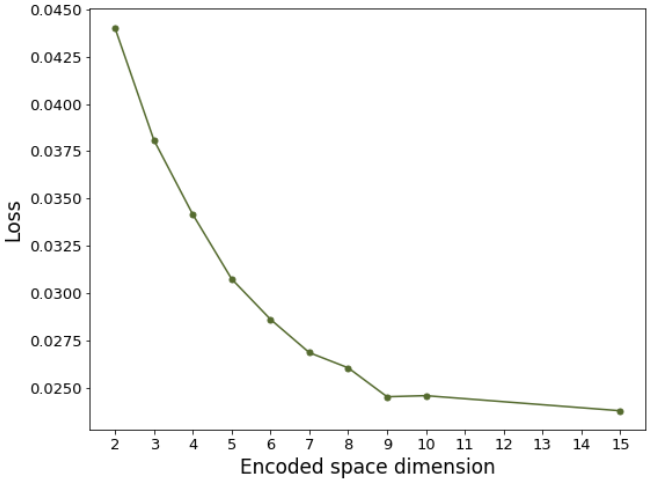

Figure 1: *Loss vs dimension of the encoded space*

2.3. Final training

At this point the encoded space dimension was chosen to be kept fixed and equal to 6. Using this value and the previously found set of hyperparameters the architecture was trained in order to save the final model. Its reconstruction error on unseen data (the test set) is *0.028*.

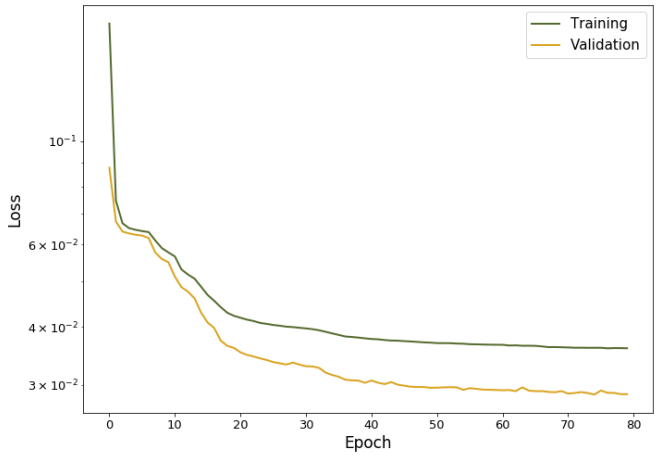

Figure 2: *Loss evolution while training the final model*

The model whose parameters are in Table 1 and with *enc_dim = 6* is saved in *Final_model.torch* and is the one that will be used in the following analysis (if not otherwise stated).

3. Corrupted data

In this section it is reported how the previously trained model is challenged using damaged data.
Two functions were implemented to modify the images in the *MNIST*: one adds normally distributed noise (with fixed mean *0* and variable standard deviation σ) to each pixel and the other deletes a squared portion p of the image (by multiplying the corresponding pixels by *0*). For the Gaussian noise it is possible to choose the value of σ whereas for the occlusion function the free parameter is the fraction of pixels to delete p.
The previously implemented test and train functions were also slighlty modified: the previous version would take in input a clean image and compute the loss over the same image and the output of the network. The new version has the possibility to use as target and input different images. This feature is fundamental to see how the autoencoder behaves when fed with damaged data.

3.1. Reconstruction capabilities

The standard autoencoder is trained on "clean" images of digits: it takes one image in input and creates its encoded version which is subsequently decoded. The procedure is successful if the final image is as similar as possible to the initial one. As mentioned at the end of Section 2 the reconstruction error of the autoencoder trained in this way is more than satisfactory, but **how** good are its reconstruction capabilities? Is it able recognize damaged images? And to which extent?
To investigate this property the autoencoder was fed (in test mode) with damaged images (modified with different levels of the same "damage") and the loss was computed using the "clean" version of the image as target.

Figure 3 shows the loss value of the model when tested on modified versions of the test set (obtained by occlusion and noising as described at the beginning of the section). As expected the performance worsens as the images becomes less and less recognizable.

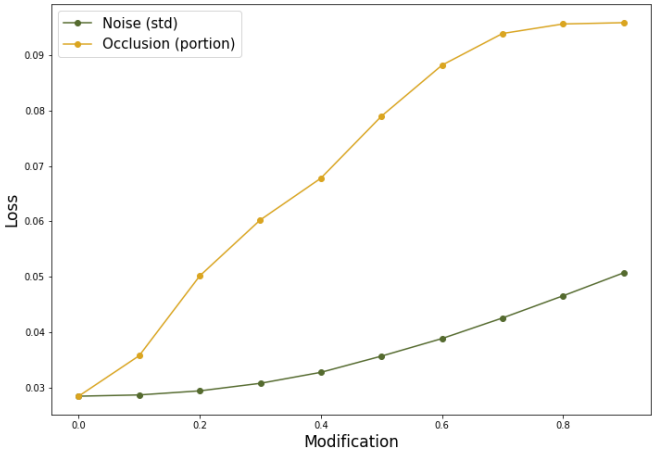

Figure 3: *Loss vs modification parameter: σ for Gaussian noise and p for occlusion*

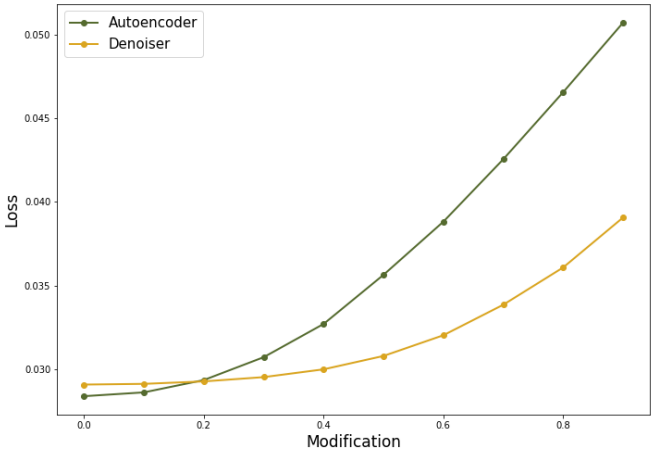

Figure 4: *Loss vs modification parameter for the basic autoencoder and a denoiser trained over a dataset with added standard noise ($\sigma = 0.45$)*

Figures 7 and 8 in the appendix show some examples of reconstruction of damaged images. It is possible to notice that, in the majority of the cases, even when the noisy digit is hardly recognizable, the decoded image corresponds to the label, even thought the final resolution is not as high. As for occlusion, this type of modification appears to impact the performance much more than noise, as Figure 3 shows.
The *MSE* loss on the noisy test set (σ=0.45) is *0.033* whereas when 20% of the image is occlude it becomes *0.05*.

3.2. Denoising

The basic reconstruction capabilities of the model are very good if compared to the loss over clean images. This feature can be improved by training the network over an already damaged dataset and updating the weights so that they minimize the loss between the reconstruction of the damaged image and the clean one. This process is called *denoising*, as the goal is to obtain a cleaned image from a noisy one.
Figure 4 shows a comparison between the losses obtained using the autoencoder and the denoiser over different noisy versions of the test set. The denoiser was trained over a noisy version of the training set with $\sigma = 0.45$ and has a *MSE* test loss of *0.0304*. As expected, this is an improvement compared to the standard autoencoder. Moreover, as Figure 4 clearly shows, the performance worsens as the noise level gets higher, but it is still way better than the simple autoencoder. Notice, however, how for low values of noise the denoiser performs slightly worse than the basic autoencoder.

4. Exploring the encoded space

In subsection 2.2 the role of the dimension of the encoded space was investigated from a performance point of view. In this section some features of the encoded space are explored to shed some light on how it works.

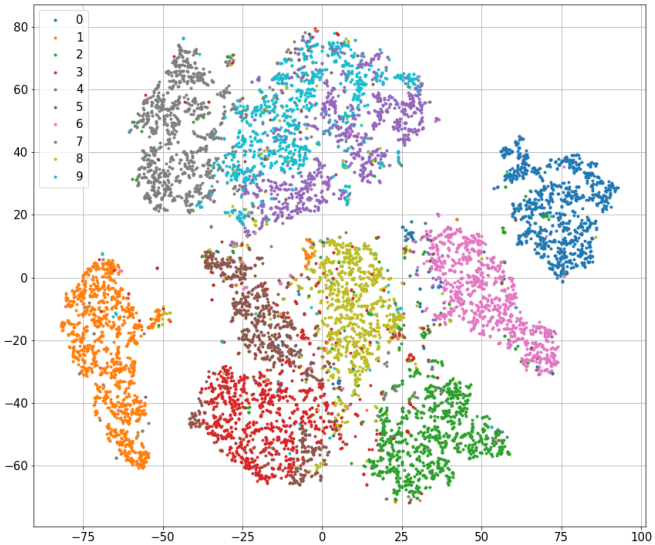

Figure 5: *t-SNE 2-dimensional representation of the encoded space*

Figure 5 shows how the encoded representation of the digits in the test set appears. Of course the encoded space is 6-dimensional, so *t-SNE* was performed to visualize it in 2 dimensions, still keeping as much of its structure as possible. It is possible to notice that equal digits are encoded in clusters i.e. nearby points in the encoded space.
Figure 6 shows the "space walk" from digit *4* to digit *9* in *10* steps (Figure 10 shows the same from

digit *6* to *9*). This image allows one to answer, from a qualitative point of view, an interesting questions: is the encoded space regular? i.e. do nearby points correspond to similar digits? Figure 6 was obtained firstly computing the centroids of the clusters of *4* and *9* i.e. feeding the network with all the test images representing the given digit, obtaining their encoded representations and computing the centroid. Once obtained the centroids for both digits, the "path" from one to the other is obtained linearly spacing their distance. At each step corresponds a new points in the encoded space that is then decoded providing the images in Figure 6. It is possible to see that the space, given this representation, looks regular indeed, and no harsh chages appear in the image form one step to the next one. This is interesting considering that the steps taken cannot be considered infinitesimally small.
Despite no quantitative measure of the smoothness of the space was computed, the images appear to change not only smoothly and regularly, but also in the most "clever" way: the natural modification to do in order to get from a *4* to a *9* is to add a bar on top and this is exactly what happens.

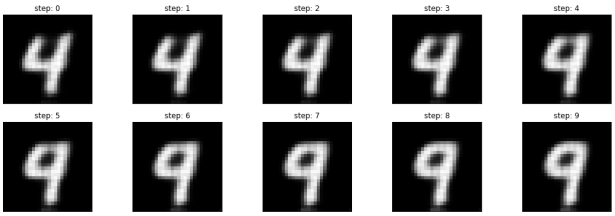

Figure 6: *Morphing from digit 4 to 9*

Appendix

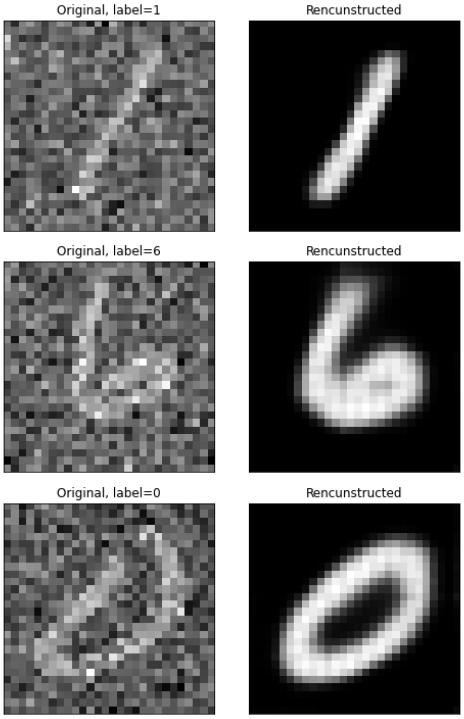

Figure 7: *Reconstruction of noisy images for* $\sigma = 0.5$

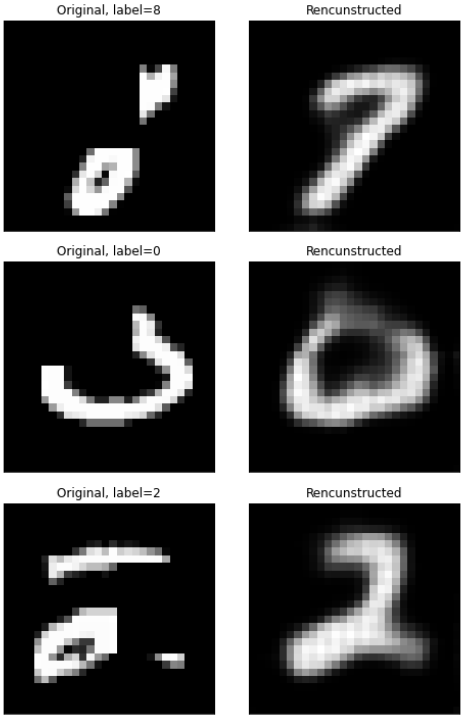

Figure 8: *Reconstruction of occluded images for p=0.2*

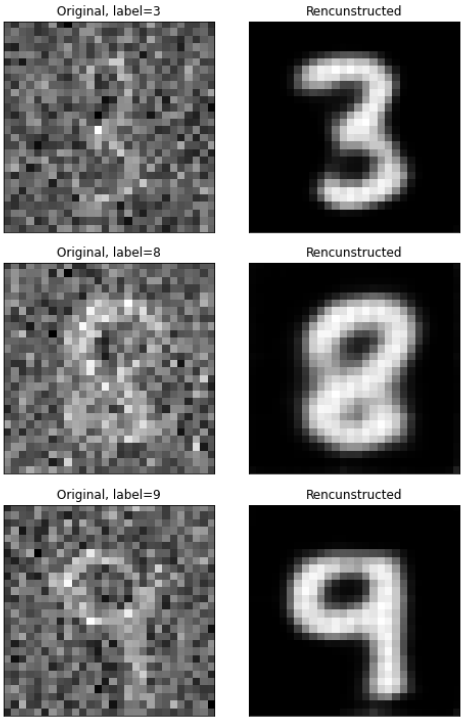

Figure 9: *Reconstruction of noisy images for* $\sigma = 0.45$ *by the denoiser of section 3.*

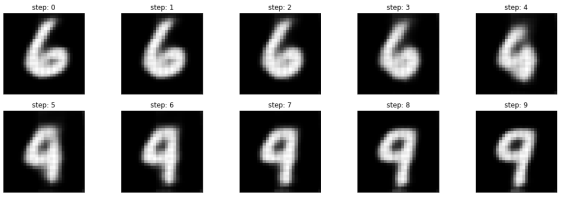

Figure 10: *Morphing from 6 to 9*