Compressing MNIST dataset with autoencoders

Nicola Dainese

I. INTRODUCTION

In this project I implemented in Pytorch an Autoencoder, tuning its parameters through a random search and training it on the MNIST dataset. I then explored the compression-performance trade-off, varying the dimension of the encoded space of the autoencoder, and testing for different dimensions the performance on the test set in three different setups: using the test set uncorrupted, adding to it gaussian noise or occluding part of the images. I then repeated the same testing procedure on denoising autoencoders. Finally I implemented a method for generating images with the Autoencoder sampling from the encoded space and analysed the smoothness and homogeneity properties of a bi-dimensional encoded space.

II. RANDOM SEARCH AND RESULTS

In this section I discuss the systematic optimization of the Hyper-Parameters (HPs) of the model.

The model considered for the encoding part had 3 convolutional layers without pooling and 2 linear layers; the decoding part was symmetric, hence it was composed by 2 linear layers and 3 transposed convolutional layers.

My approach to HP's optimization was to reuse the code for the random search with prior distributions implemented and discussed during the second assignment. I decided to optimize all the HPs except from the dimension of the encoded space, fixing the last to an intermediate size of 4. This was because in the second part of this work I will study the dependence of performance of the Autoencoder on this parameter. The random search works as follows: for each HP I define a prior distribution to encode my beliefs on the best choice of parameters; this can be uniform or logarithmic in a given interval for continuous variables, or a custom distribution for discrete variables. I then decide how many configuration to test and draw independently from each prior that exact number of parameters and train the model on each configuration of HPs.

The HPs considered were the following:

1. Learning rate: logarithmic prior (to sample uniformly at different scales) in $[10^{-1}, 10^{-3}]$.
2. L2 penalty: logarithmic prior in $[10^{-2}, 10^{-4}]$.
3. Dropout: uniform in $[0, 0.25]$.
4. Number of epochs of training: values [10,20,30,40,50] with probabilities [0.25, 0.25, 0.20, 0.20, 0.10].

In addition to this, I made other two choices:

1. I used Adamax as optimizer, mainly because of its stability with different learning rates.
2. I used a simple 80-20 split in training and validation sets from the original 60.000 samples of the MNIST training set; this is because I also tried K-fold cross validation, but there was a negligible difference between the validation loss in the different folds, so it wasn't convenient.

During the random search I used Google Colab to train and validate almost 180 configurations thanks to the GPU at disposal.

The results weren't much interesting, since I found no dependence of the performance on learning rate, dropout and number of training epochs. The only clear result was that the loss increases monotonically with the L2 penalty, as can be seen in Figure 1.

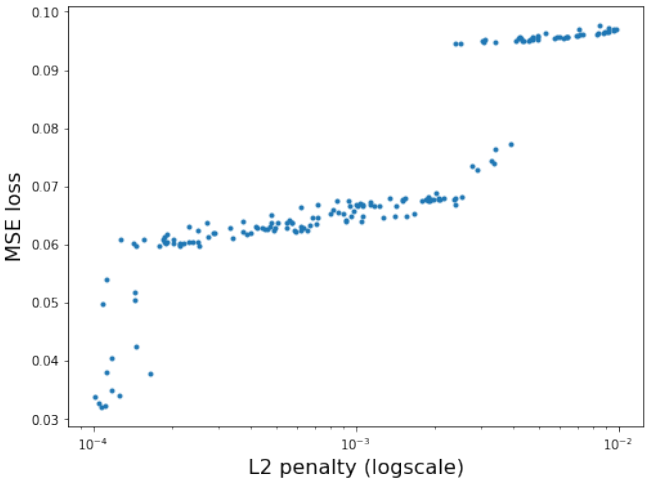

FIG. 1: Dependence of the Mean Square Error loss on the L2 penalty.

For the sake of completeness I report in the table all the HPs of the most performing configuration on the validation set.

Hyper-parameters	
HP	Value
Dropout	0.022
Number of epochs	40
Learning rate	0.013
Penalty weight	10^{-4}

III. COMPRESSION-PERFORMANCE TRADE-OFF

To test how the performance is affected by the dimension of the encoded space I evaluated different models trained with the same HPs but with the following dimensions of the encoded space: [1,2,4,8,16]. Tests of the performance were made on 3 different kinds of versions of the test dataset: the original test set, the same set with the addition of gaussian noise of different intensities and again the same set with part of each image occluded (different scales of occlusion were tested). In Figure 2 can be seen that the loss of the Autoencoder on the uncorrupted test set decreases monotonically with the dimension of the encoded space, as we could have expected. In Figure 3 instead we can see that for higher dimensions of the encoded space there is a greater loss in performance as the noise increases, whereas if the dimension is 1 or 2 the performance is quite stable and robust to noise. Finally in Figure 4 we can see a similar trend as the portion of the input image occluded increases, but this time the trend is not smooth.

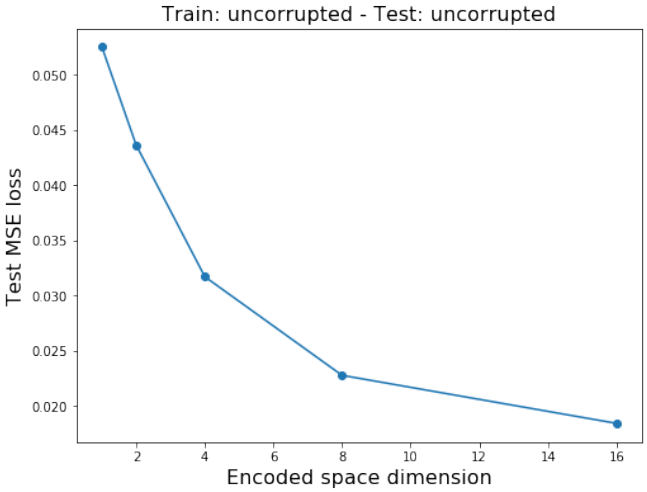

FIG. 2: Dependence of the Mean Square Error loss on the dimension of the encoded space.

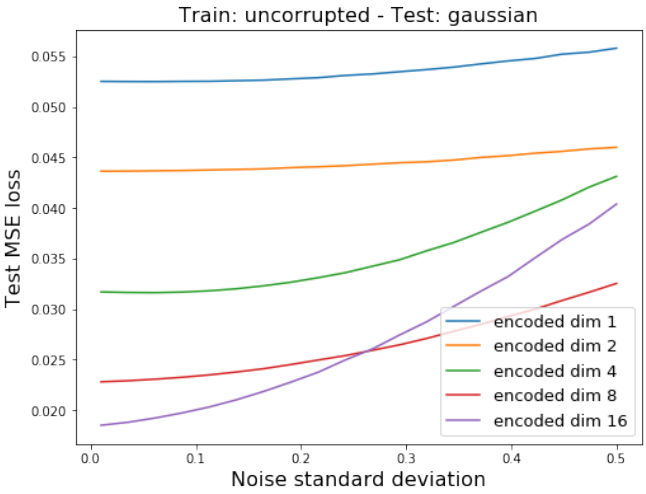

FIG. 3: Dependence of the Mean Square Error loss on the intensity of the gaussian noise for different dimensions of the encoded space.

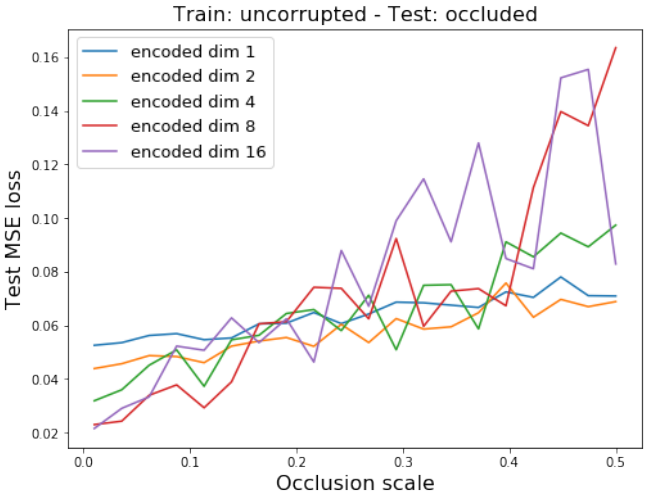

FIG. 4: Dependence of the Mean Square Error loss on the linear scale of the occlusion for different dimensions of the encoded space.

IV. DENOISING AUTOENCODERS

Denoising Autoencoders do not differ in architecture from standard Autoencoders. The only difference is in how they are trained: this time, instead of giving them as inputs uncorrupted images, we will feed them noisy or occluded images and as targets the uncorrupted images, so that they will learn how to denoise an image or to reconstruct a missing piece. I reproduced the same results of the previous section with denoising Autoencoders, but the results were qualitatively identical and slightly better quantitatively. For these reasons I do not report the results here, but they can be seen in the Analysis notebook.

V. SMOOTHNESS AND HOMOGENEITY PROPERTIES OF THE ENCODED SPACE

In this section firstly I introduce a sampling technique from the encoded space of an Autoencoder and then use it to inquire the smoothness and homogeneity properties of that space.

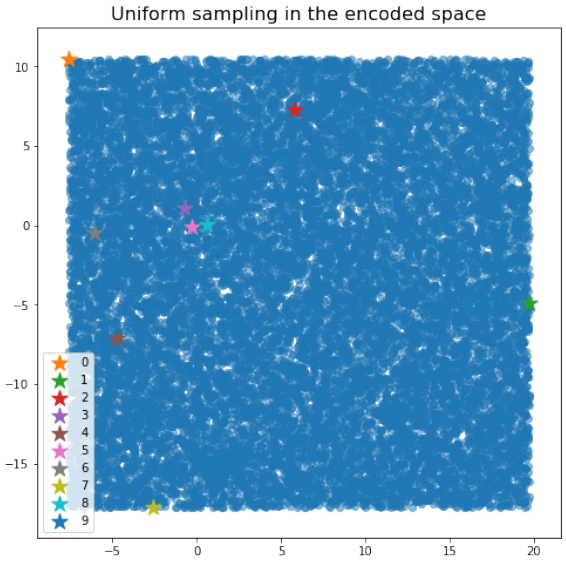

FIG. 5: Example of sampling in bi-dimensional encoded space. Stars represent the centroids of the 10 classes of digits.

A. Sampling from the encoded space

The idea of sampling from the encoded space is fairly simple: it's sufficient to draw a random number in the correct range for each dimension of the space and then form a vector from this random numbers and decode it with the second part of the Autoencoder's architecture. So the question is how to get the right range for sampling.

First of all it should be noticed that, since there is no softmax activation function for the final layer of the encoding architecture, the inputs are mapped in a portion of $\mathbb{R}^d$ and not just in $[0,1]^d$. As a consequence the only reasonable elements that can be used to delimit the region of space where the most interesting samples lie are the centroids of each class of digits. These are obtained computing the mean of all encoded vectors of the input images class by class. My sampling choice was to get the maximum and minimum coordinates for each dimension among the 10 centroids, to sample random vectors uniformly in that hyper-parallelepiped and then decode them to get the corresponding images.

B. Properties of bi-dimensional encoded space

As mentioned before, there are 2 interesting properties that we can study for an encoded space:

1. Smoothness: a small movement in the encoded space translates in a small change in the output space.
2. Homogeneity: a point that in the encoded space is equidistant from 2 other points, should be equidistant to them also in the output space.

To test the smoothness and the homogeneity of the encoded space I defined a function D(x,y) that is equal to the square distance in the output space from the image of the nearest centroid in the encoded space.

For example if I take as x the centroid of the digit 1, the nearest centroid y is the same as the point itself, so its function D(x,y) will be 0. Now if we take the second nearest centroid (say 7) and we move toward it, the function D will increase because we are changing the output and the distance is non-negative (ideally the output will change smoothly from a 1's image to a 7's image). When the point selected is halfway between the two nearest centroids we naturally reach a maximum, but it can be continuous or discontinuous, depending on whether the distance in the output space from the two centroids is the same or not.

Guided by this idea I realised an interactive plot with the plotly library of the surface obtained plotting the function D(x,y) by sampling with a grid scheme 10^4 points. In Figures 6 and 7 is possible to see the plot from a frontal and vertical view. Orange dots represent the 10 centroids . Visualizing the plot in the notebook is possible to hover over them to see which digit corresponds to each of them and also to rotate the plot along the three axis interactively.

In the first plot is possible to see that the space is smooth around the centroids, but has sharp transitions where we change basin of attraction (i.e. changes the centroid w.r.t. which we compute the distance). The fact that there are discontinuities in the surface means that the space is not homogeneous, because a given distance in the encoded space in general is not mapped to the same distance in the output space. This means that we cannot measure correctly the similarity between images associated to encoded vectors using the Euclidean distance in the encoded space.

In the second plot instead it's possible to see the basins of attraction of all the different digits, delimited by the ridges. From the notebook we can see that the centroids of 4 and 9 are mapped adjacent to one another and so are 3, 5 and 8. Also it's possible to see that the 2D encoded space of an Autoencoder is smooth but not homogeneous, since ridges are not smooth.

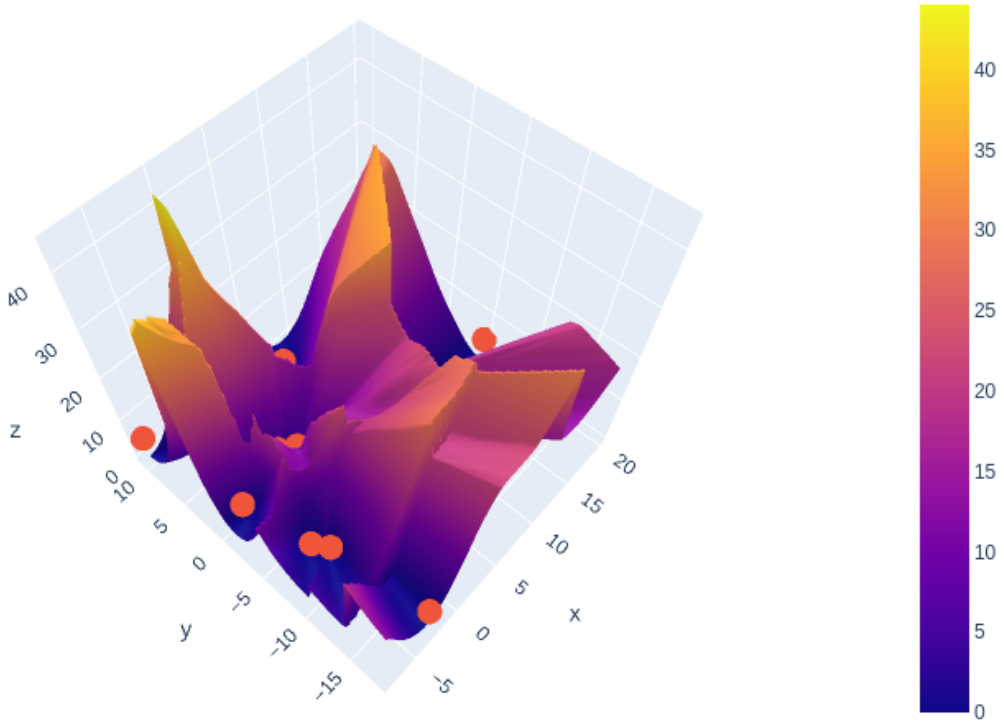

FIG. 6: 3D plot of D(x,y). Orange dots represent the centroids of the 10 classes.

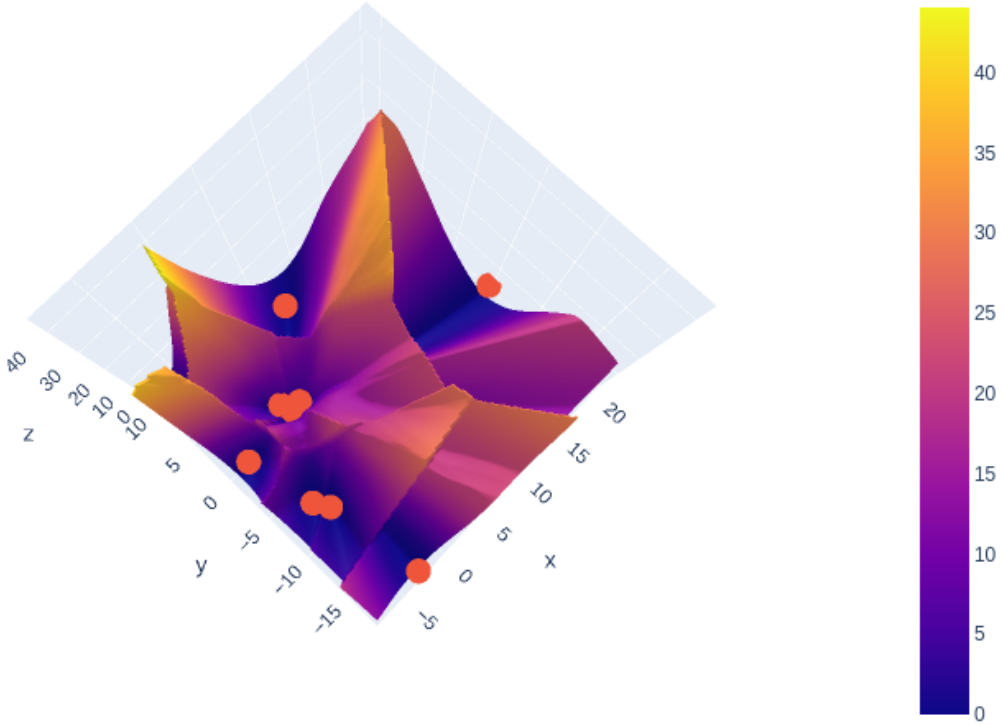

FIG. 7: Vertical view of the surface D(x,y). Orange dots represent the centroids of the 10 classes.