Homework 2: Unsupervised Learning

Neural Networks and Deep Learning

Cracco Gianmarco

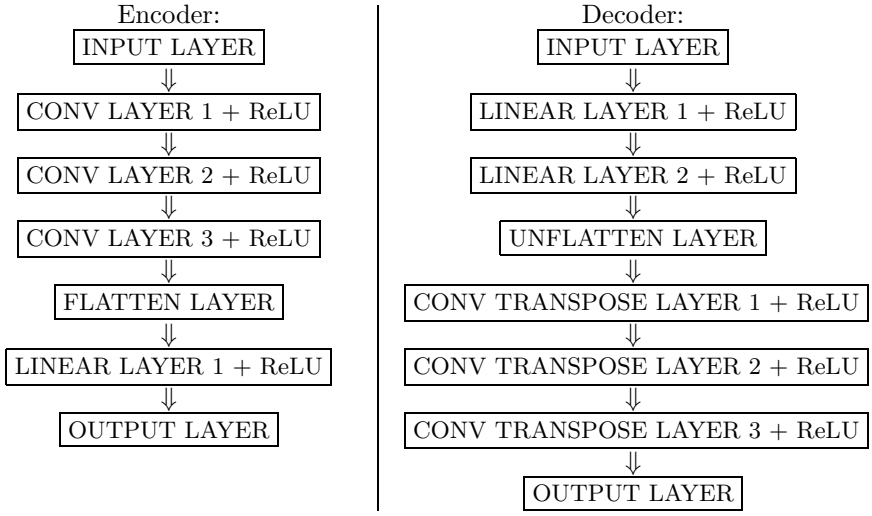
December 2020

Introduction

The goal for this homework will be centered around unsupervised learning and MNIST dataset: as first step we will implement a (convolutional) autoencoder. Once this is done we will explore its latent space; doing so we will be able see if the encoder have learned to correctly classify the input data without even looking at the target label. To make further use of the information stored in the latent space, the decoder will be detached in favour of few linear layers to classify MNIST's digits. Finally with basically the same architecture, a denoising autoencoder will be implemented.
At the end, we will implement a more advanced model: a Generative Adversarial Network (GAN) to generate new images similar to MNIST.

1 Autoencoder

First we start describing the architecture for both networks:



where the hyperparameters related to the convolutional are the following:

- Conv layer 1: `Conv2d(in channels=1, out channels=8, kernel size=3, stride=2, padding=1)`;
- Conv layer 2: `Conv2d(in channels=8, out channels=16, kernel size=3, stride=2, padding=1)`;
- Conv layer 3: `Conv2d(in channels=16, out channels=32, kernel size=3, stride=2, padding=0)`;
- Conv transpose layer 1: `ConvTranspose2d(in channels=32, out channels=16, kernel size=3, stride=2, padding=0)`;
- Conv transpose layer 2: `ConvTranspose2d(in channels=16, out channels=8, kernel size=3, stride=2, padding=1)`;
- Conv transpose layer 3: `ConvTranspose2d(in channels=8, out channels=1, kernel size=3, stride=2, padding=1)`.

1.1 Hyperparameters tuning

To train this first model the training set has been splitted into train and validation set (80%-20%), then to find the optimal combination of hyperparameters a random search has been implemented. The chosen loss function is Mean Square Error, while all the weights have been initialized with Xavier distribution[1]. Hyperparameters exploration has been carried out on 50 combinations randomly picked among the following:

- Encoded space dimension: `[a for a in range(2,10)]`;
- L2 norm: `[0] + list(np.random.normal(mean=1e-5, std=0.0001, size=10))`;
- Optimizer: `[Adam, RMSprop]`;
- Learning rate: `list(np.random.normal(mean=0.001, std=0.001, size=10))`;
- Number of units for 1^{st} linear layer(both for encoder and decoder)[2]: `list(np.random.randint(50, 150, size=10))`.

As expected the model obtains better results with the highest encoded space dimension: this is due to the fact that having more dimensions allows it to encode more information correctly. In Table 1 it is possible to see the results of the 5 best combinations.

[1] The same procedure will be applied for the denoising autoencoder and the classificator.

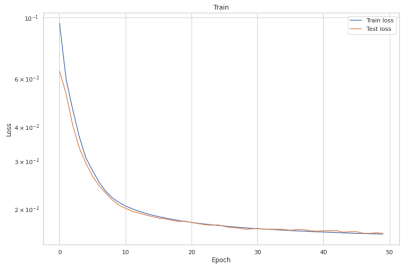
[2] the number of units for the second linear layer of the decoder has been kept fixed at 288 neurons.

Enc. Space Dim.	L2	Optimizer	Epochs	Learn. Rate	Nh_1	Valid_loss
9	5.1e-05	Adam	50	0.0009	87	0.0172
8	5.0e-05	Adam	50	0.0019	122	0.0174
9	3.2e-07	RMSprop	50	0.0019	75	0.0181
8	3.2e-07	RMSprop	50	0.0019	137	0.0182
9	3.2e-07	Adam	50	0.0003	137	0.0183

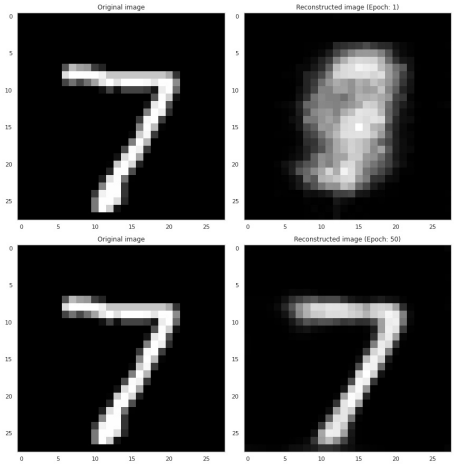
Table 1: Results of first 5 best models.

1.2 Final Train

With the best configuration of hyperparameters, a final train has been executed on all training set achieving a MSE = 0.016 on the test set:



(a) Final train losses.



(b) Reconstructed image at the beginning and at the end of the training.

2 Latent Space Exploration

To be able to explore its latent space (encoded coordinates of 5 samples are reported in Table 5), 2 of the most famous techniques of dimensional reduction has been implemented: PCA and t-SNE. However only the latter has been able to give a meaningful plot of the encoded data reducing the dimension from 9 to 2. The plot obtained is reported here in Fig. 2

As we can see the encoder has been able to correctly clusterize the images according to their labels without having them as input.
Picking random 9d coordinates it is possible to generate new samples: to obtain Fig. 3, an encoded sample has been taken as starting point and then before feeding it to the decoder a random gaussian noise ($\mu = 5$, $\sigma = 10$) has been

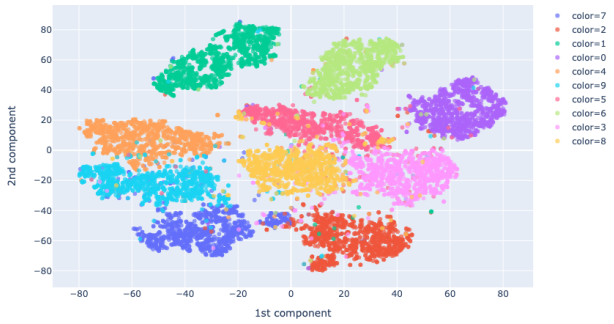


Figure 2: t-SNE representation of the encoded space.

applied in order to obtain 16 different new images.

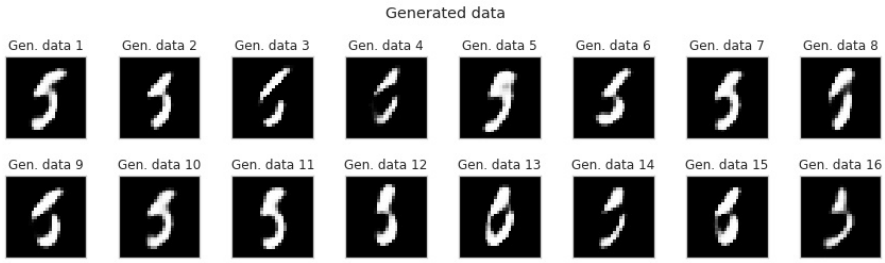


Figure 3: New images obtained applying random gaussian noise starting from encoded sample.

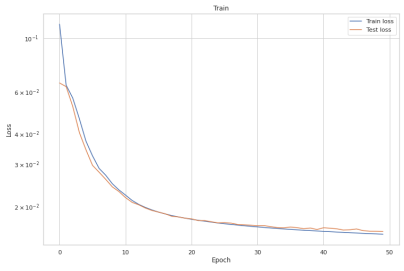
Additional trials with PCA (2d & 3d) and t-SNE (3d) are reported in Appendix A.1 in Fig. 8, Fig. 9 and Fig. 10 respectively.

3 Denoising Autoencoder

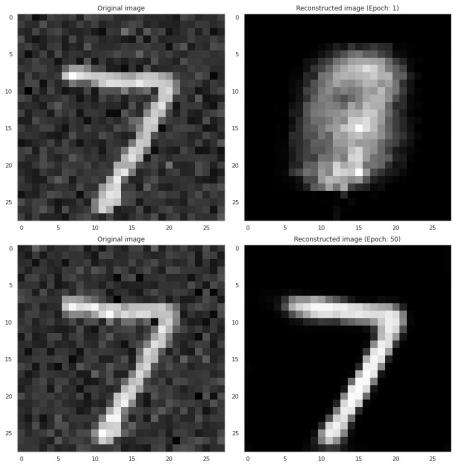
To complete this task the entire model and training procedure has been kept identical as before, the only difference is that as input of the encoder, a noisy image is provided and then the loss in computed comparing the decoder output with the original ones.
To obtain a noisy image a simple gaussian noise has been added to the original data ($\mu = 0$, $\sigma = 0.1$). Proceeding in the same way as in Section 1.1 and 1.2, in Fig. 4 is reported the losses plot and the reconstructed data at the beginning and at the end of training:

Enc. Space Dim.	L2	Optimizer	Epochs	Learn. Rate	Nh_1	Valid_loss
9	1.5e-06	Adam	50	1.0e-03	137	0.015
9	4.2e-06	Adam	50	1.9e-03	64	0.017
8	5.1e-06	Adam	50	1.8e-03	136	0.017
9	1.5e-06	RMSprop	50	1.6e-03	137	0.018
9	4.2e-06	RMSprop	50	1.8e-03	144	0.020

Table 2: Results of first 5 best models.



(a) Final train losses.

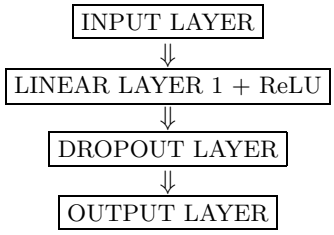


(b) Reconstructed image at the beginning and at the end of the training.

Figure 4: MSE = 0.16.

4 Autoencoder for Classification task

The goal of this task was to implement a convolutional autoencoder for supervised classification task.
Starting from the trained encoder obtained in Sec. 1, the decoder has been detached and replaced with the following architecture:



Where the input layer of course has dim=9 while the output one has dim=10 (like the number of classes).
Using the Cross Entropy as loss, a grid search has been implemented for 20

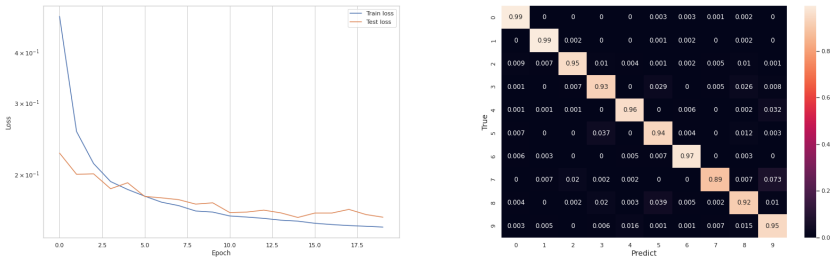
epochs for each combination, picking the hyperparameters from the following lists:

- Number of units for 1^{st} linear layer: `list(np.random.randint(50, 300, size=6))`;
- L2 norm: `[0, 1e-03, 1e-05]`;
- Optimizer: `[Adam, RMSprop]`;
- Learning rate: `[0.001, 0.005]`;
- Dropout probability: `[0, 0.05, 0.1]`.

Proceeding in the same way as in Sec. 1.1 and Sec. 1.2:

Nh1	Optimizer	Epochs	Lear. Rate	Dropout p	Weight_dec	Valid_loss
184	RMSprop	20	0.001	0.10	1e-05	0.147
227	Adam	20	0.001	0.05	1e-05	0.148
227	Adam	20	0.001	0.05	0	0.148
227	RMSprop	20	0.001	0.10	0	0.149
227	Adam	20	0.001	0.10	0	0.149

Table 3: Results of first 5 best models.



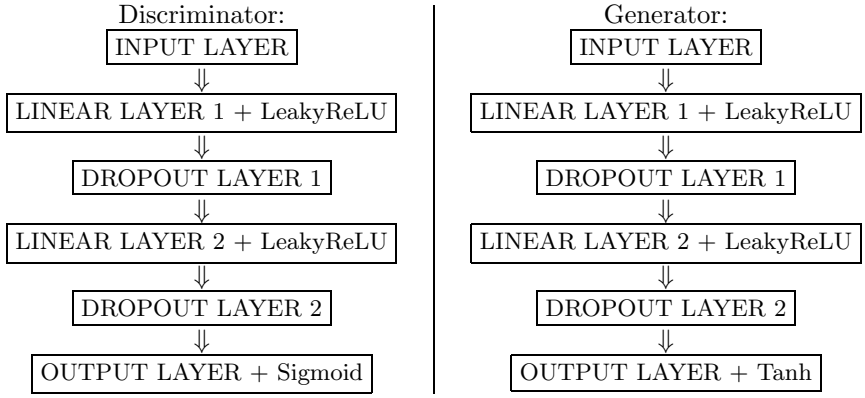
(a) Final train losses. (b) Confusion matrix.

Figure 5: Accuracy on test set = 94.85%

As we can see the accuracy it's not so high comparing it with what has been obtained in the previous Homework ($\sim$ 97.86% -ed.). However we can notice how the number of epochs of this training has been lowered from 100 to 20, hence reducing by a factor $1/5$ the time required.

5 GAN

As first step the input data have been normalized between $[-1, 1]$, then the two networks were implemented as following:



where the hypeparameters related to both networks are reported in Table 4:

	Input dim.	Hidden dim. 1	Hidden dim. 2	Output dim.	LReLU slope	Dropout p	Optimizer	Lr
Discr.	784	512	256	1	0.2	0.3	Adam	5e-4
Gen.	100	256	512	784	0.2	0.3	Adam	2e-4

Table 4: GAN Hyperparameters.

Using Binary Cross Entropy as loss, the whole model has been trained for 400 epochs: 256 100d-vectors randomly sampled from standard normal distribution (i.e. $X \sim Unif(\sqrt{100} \cdot S^{99})$, where S^n is a $n-$dimensional unit sphere) are fed to the generator as input; while as a training routine, instead of feeding to the discriminator both real and generated images at the same time, small batch (256 samples) of real or generated were used alternatively.
By doing so, it has been possible to correctly sync the learning procedure of both networks. The obtained results are reported in Fig. 6 and Fig. 7, while the final loss reached by the networks are respectively:

- Discriminator loss: 0.81;
- Generator loss: 1.20.

Animated version of Fig. 7 and more images related both to this Section and the previous ones are available here.

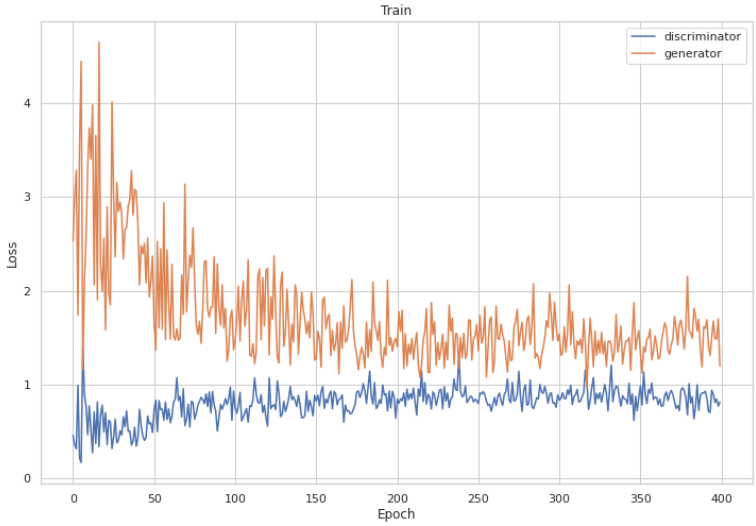


Figure 6: Loss of generator and discriminator during training.

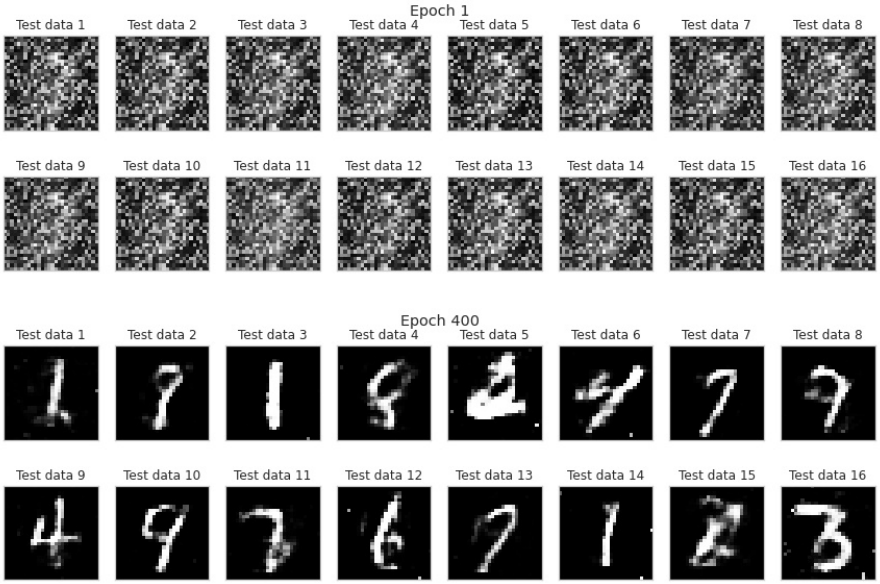


Figure 7: Generated data at the beginning and at the end of training.

6 Conclusions

All tasks have been accomplished correctly. Further improvements can be made by exploring more complicated architectures or more sophisticated optimization procedures like bayesian optimization. For what concerns denoising autoencoder, more attempts could be made using other types of noise (like salt and pepper or noise distributed in a different way from the Gaussian one) or even try to teach an autoencoder to remove all of them instead of training with only one at the time.
To improve GAN instead, an obvious modification could be implementing (de)CNNs as generator and/or discriminator and performing hyperparameters optimization.

A Appendix

A.1 Latent Space

	Enc. Variable 1	Enc. Variable 2	Enc. Variable 3	Enc. Variable 4	Enc. Variable 5	Enc. Variable 6	Enc. Variable 7	Enc. Variable 8	Enc. Variable 9	label
0	8.353384	6.062208	6.882480	-23.443331	-41.030186	15.353236	-19.266520	8.581726	13.493169	7
1	0.835941	-20.711342	5.206129	-45.665855	16.590822	27.288141	11.172687	-24.107357	-23.720579	2
2	12.370504	-6.326145	25.866806	-14.534688	2.658181	21.562832	7.883648	2.671305	17.432297	1
3	0.243718	16.727535	-5.982775	-50.432438	-26.678438	12.678731	33.965645	-3.363263	6.581146	0
4	-24.115749	-3.651129	17.732262	-16.592127	-28.590008	-0.602595	20.050447	-10.631841	10.671299	4

Table 5: First 5 encoded samples.

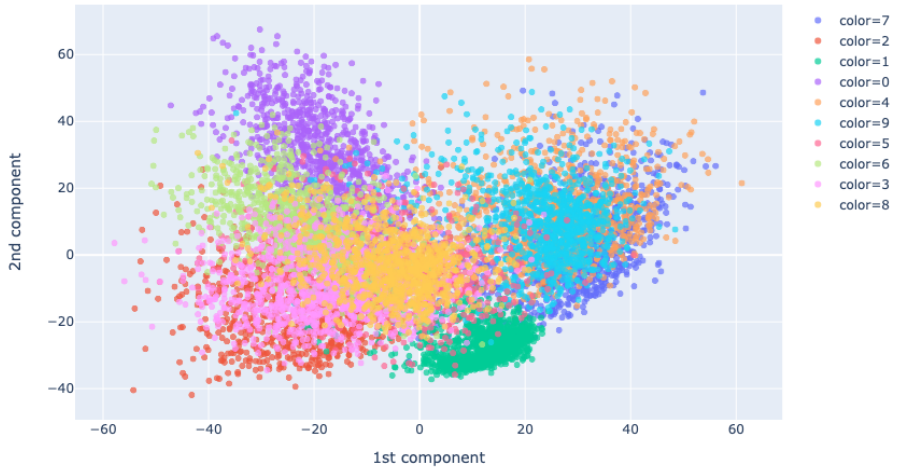


Figure 8: Latent space representation using PCA and 2 components.

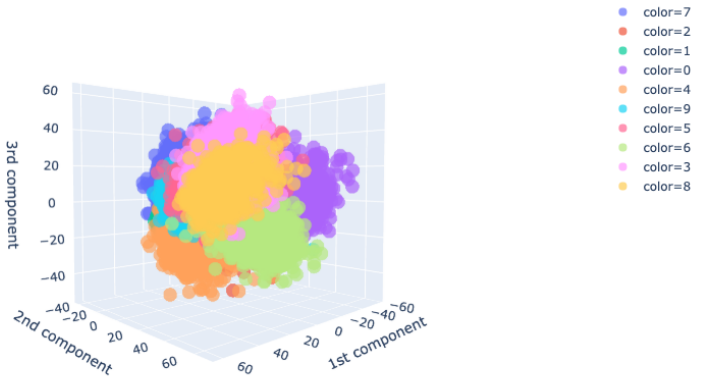


Figure 9: Latent space representation using PCA and 3 components.

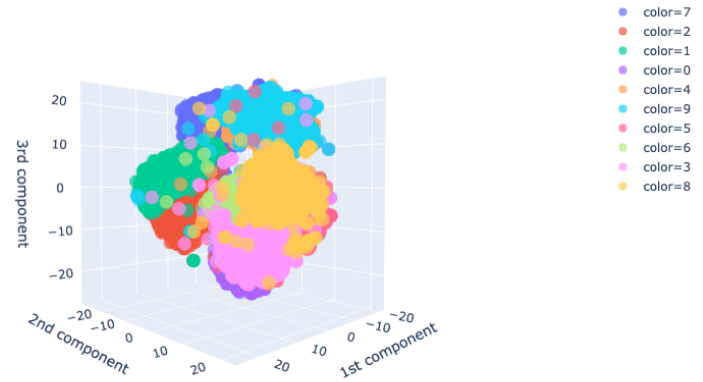


Figure 10: Latent space representation using t-SNE and 3 components.