Homework 2 - Unsupervised Deep Learning

Michele Puppin ID 1227474 michele.puppin@studenti.unipd.it

(Dated: August 26, 2021)

I. INTRODUCTION

In this homework we will implement and test simple neural network models for solving unsupervised problems, where the network is asked to learn an internal representation of the environment that can be later exploited for other learning porpouses. Firstly, we will test and analyze a convolutional autoencoder over a dataset of images of handwritten digits (MNIST). In particular, advanced optimizers and regularization methods will be implemented and learning hyperparameters will be tuned using appropriate search procedures. Moreover we will implement denoising convolutional autoencoder, where some noise is added to the input image and the network is trained to remove the noise. Then, we fine-tune the convolutional autoencoder using a supervised classification task, we explore the latent space structure and generate new samples from latent codes. Finally, we implement variational autoencoder which are analogous model based on a probabilistic approach.

These models are implemented in Python using the PyTorch framework.

II. CONVOLUTIONAL AUTOENCODERS

1. Methods

Autoencoders are a specific type of feed-forward neural networks composed by two main parts: an encoder that maps the input into the code, and a decoder that maps the code to a reconstruction of the input. The code is called the latent-space representation. Autoencoders are forced to reconstruct the input approximately (an identity mapping would be useless), preserving only the most relevant aspects of the data in the copy. Therefore they are useful for dimensionality reduction or feature learning. Autoencoder training is performed through Back-Propagation algorithm, by comparing the original input to the reconstructed one.

The architecture chosen for the encoder is the following:

- Convolutional part:
  - a first convolutional layer with one input channel (the image is in gray scale) and 8 output channels, we use a 3×3 filter that slides over the image with a stride of 2 and a padding of 1;
  - a second convolutional layer with 8 input channels and 16 output channels, we use a 3×3 kernel with a stride of 2 and a padding of 1;
  - a third convolutional layer with 16 input channels and 32 output channels, we use a 3×3 kernel with a stride of 1 and a padding of 0;
- Linear part:
  - a first linear layer with an input size of $(32 \times 3 \times 3)$ and an output size of 64;

– a second linear layer with an input size of 64 and an output size which is a variable (*encoded space dimension*) and will be tuned with model selection.

The output of the convolutional part is flattened before being passed to the linear part. The decoder architecture is exactly symmetrical.

Moreover, we use ReLU activation function, we use Adam optimizer and Mean Squared Error (MSE) loss function.

Again, we use the optimization techniques introduced in *Homework 1* and we use a random grid search to tune the hyperparameters, which are the following:

- the learning rate $\eta \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$;
- the L2 regularization parameter $\lambda \in \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$;
- finally, the encoded space dimension between $4, 8, 16$.

Once we find the best performing hyperparameters, we run the training procedure over the whole training set and then we test the performances over the test set.

2. Results

The best parameters resulting from model selection using random grid search are:

- the learning rate $\eta = 10^{-3}$;
- the L2 regularization parameter $\lambda = 10^{-5}$;
- the encoded space dimension is 16.

Training and validation profile for training with best hyperparameters is plotted in Fig. 1a. Early-stopping procedure stops the training at around 85 epochs. The trained model is test over the test dataset and we get a test error of 0.0093. Some example of image reconstruction are shown in Fig. 1b. Both from a quantitative and a qualitative point of view, we observe very good performances.

In conclusion, the network architecture chosen is able to perform learn to reconstruct the input images with a low error.

III. DENOISING CONVOLUTIONAL AUTOENCODERS

1. Methods

Denoising autoencoders are models that take a partially corrupted input and are trained to recover the original undistorted input. To this scope we use the same architecture presented above and we train the network by providing the input with the addition of a gaussian noise.

2. Results

We directly train the model with the best hyperparameters found above. Training and validation profile for training with best hyperparameters is plotted in Fig. 2a. Early-stopping procedure stops the training at around 90 epochs. The trained model is test over the test dataset and we get a test error of 0.0094. Some example of image reconstruction are shown in Fig. 2a. We can see that despite the noise addition, the model is still able to reconstruct the input image with good performaces.

In conclusion, we see that convolutional autoencoders perform well as denoisers.

IV. FINE TUNING WITH SUPERVISED CLASSIFICATION

1. Methods

Now we exploit the latent-space representation produced by the pre-trained encoder for a classification task. To do that we add after the encoder a linear readout part composed by a fully-connected layer with 64 neuron and an output layer with 10 neurons (one for each of the possible classes). The network will be trained to classify the digits, only readout layer weight are updated, while encoder weights remain unchanged during the training. Log-softmax activation function is used in the output layer. We use Adam optimizer with a learning rate of 10^{-3} and the L2 regularization parameter set to 10^{-5}. Negative log-likelihood loss function is used. This strategy is computationally convenient since we need to train only the readout part parameters.

2. Results

Training and validation profile for training is plotted in Fig. 3a. Early-stopping procedure stops the training before before 20 epochs. The resulting accuracy measured on the test dataset is 0.9582 and the confusion matrix obtained is shown in Fig. 3b

Considering classification performance of the network architecture implemented in *Homework 1* we can see that the accuracy is lower for the autoencoder (it was 0.9927 for the feed-forward neural network) but convergence is reached in around 20 epochs (compared to the 60 needed for the feed-forward neural network).

V. LATENT SPACE STRUCTURE ANALYSIS

1. Methods

The encoder embeds the features in the latent space. The information encoded in such a way in the latent space can be analyzed. Firstly, we use Principal Component Analysis, a technique for reducing the dimensionality of a datasets, increasing interpretability but at the same time minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance. Secondly, we use the t-distributed stochastic neighbor embedding (t-SNE) which is a technique for dimensionality reduction which preserves the dataset local structure and it is particularly well suited for the visualization of high-dimensional datasets. The core point is to associate to each data point a Gaussian probability distribution for its neighborhood. Eventually, some gaussian noise is provided as input to the decoder and new samples are generated.

We use the model trained as in the first section and we analyse the outputs of the encoder for the test dataset.

2. Results

In Fig. 4a the latent space scatter plot is shown, as expected its structure is pretty chaotic. Scatter plot for PCA analysis is shown in Fig. 4b; PCA proves not to be effective in this case, clusters relative to each digit are not clearly distinguishable. In Fig. 4c, instead, we can see that tSNE algorithm performs well clustering significantly each digit.

Some examples of new generated samples are shown in Fig. 5. No shape can be recognized.

VI. VARIATIONAL AUTOENCODERS

1. Methods

Autoencoder latent space has shown to be irregular, so a small variation of the input can produce very different outputs. To overcome this problem we can use Variational Autoencoder, models that use a probabilistic approach. By reparametrizing the sampled latent representation appropriately Back-Propagation algorithm can be used to train the network.

The architecture used for the encoder is:

- an input layer with 748 neurons;

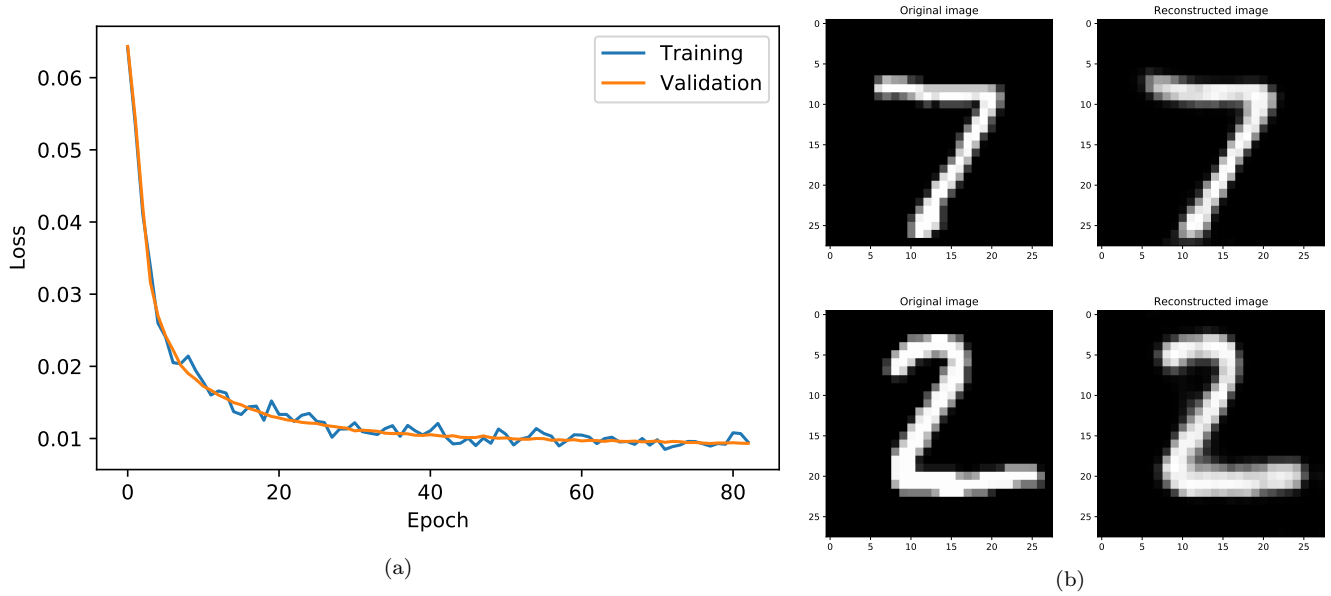
- a first hidden layer with 512 neurons;
- a second hidden layer with 256 neurons;
- the mean coding layer with 2 neurons;
- the variance coding layer with 2 neurons;

The decoder architecture is exactly symmetrical. The model is trained over the whole training set. Some new samples are generated providing noise as input to the decoder.

2. Results

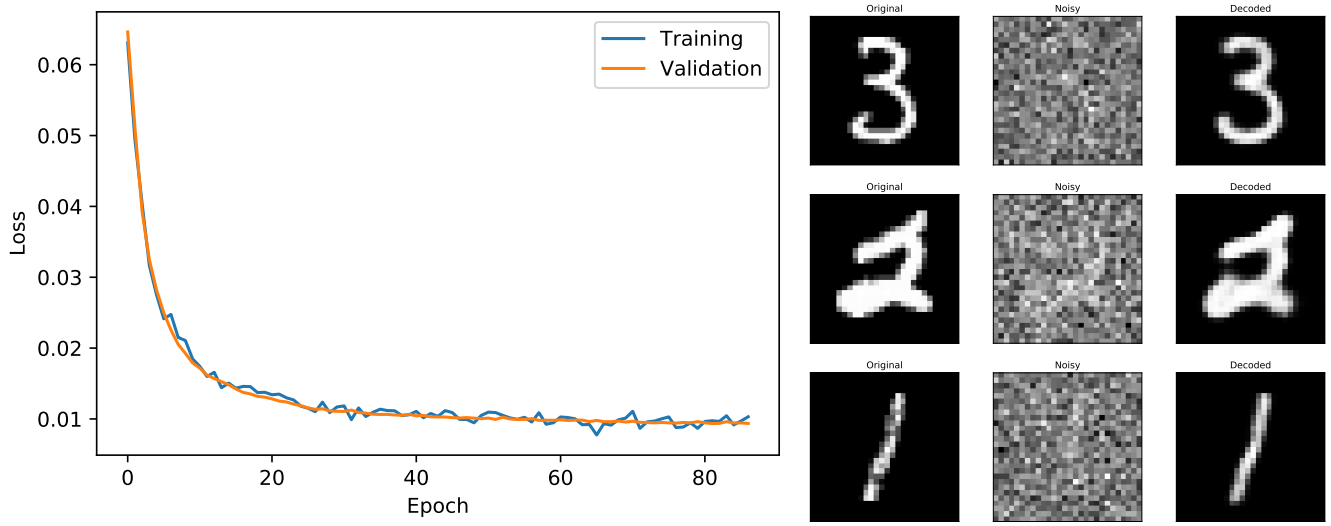
The model is trained for 50 epochs. Training and validation profile for training is plotted in Fig. **6**a. In Fig. **6**b we can see that samples generate by a variation convolutional autoencoder are much clearer and easily distinguishable.

VII. APPENDIX



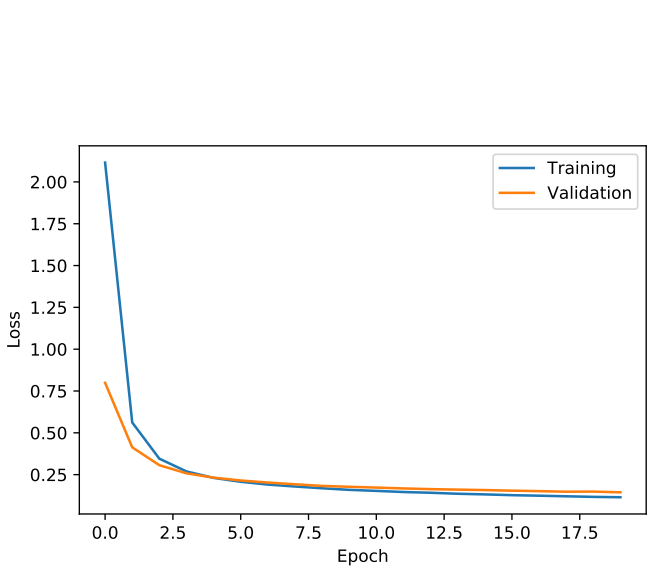
(a) (b)

FIG. 1 Convolutional Autoencoders. (a) Training and validation loss profile for training epochs. (b) Examples of image reconstruction.

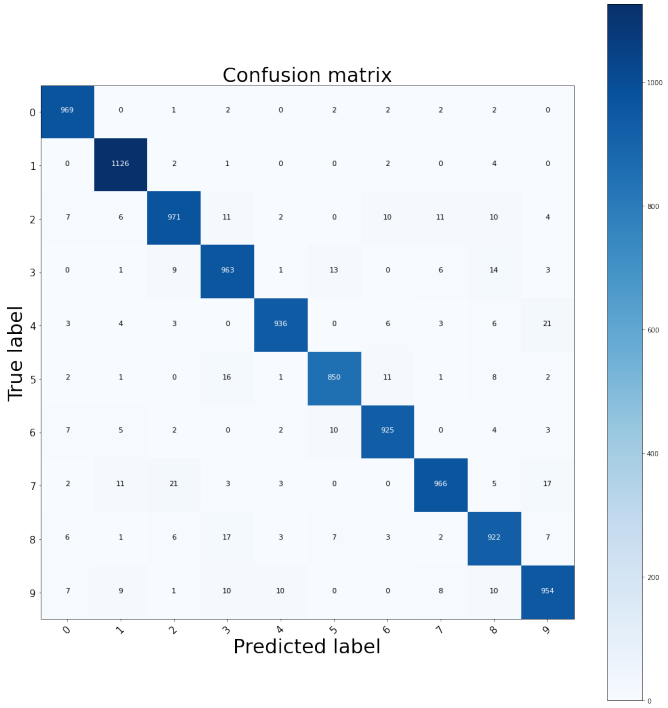


(a) (b)

FIG. 2 Denoising Convolutional Autoencoders. (a) Training and validation loss profile for training epochs. (b) Examples of image reconstruction.

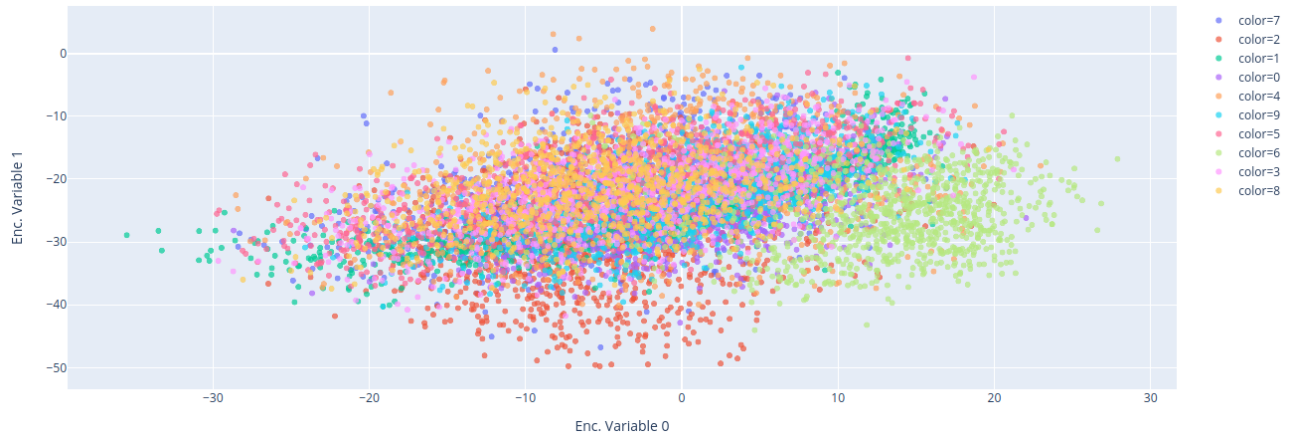


(a)



(b)

FIG. 3 Fine tuning with supervised classification. (a) Training and validation loss profile for training epochs. (b) Confusion matrix for the test dataset.



(a)

(b)

(c)

FIG. 4 Latent space structure analysis. (a) Scatter of encoded space of test samples. (b) Scatter PCA analysis of encoded space of test samples. (c) Scatter tSNE analysis of encoded space of test samples.

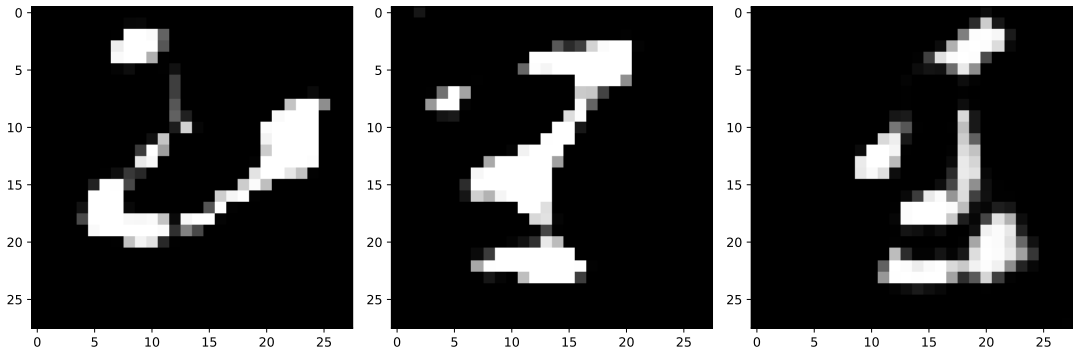
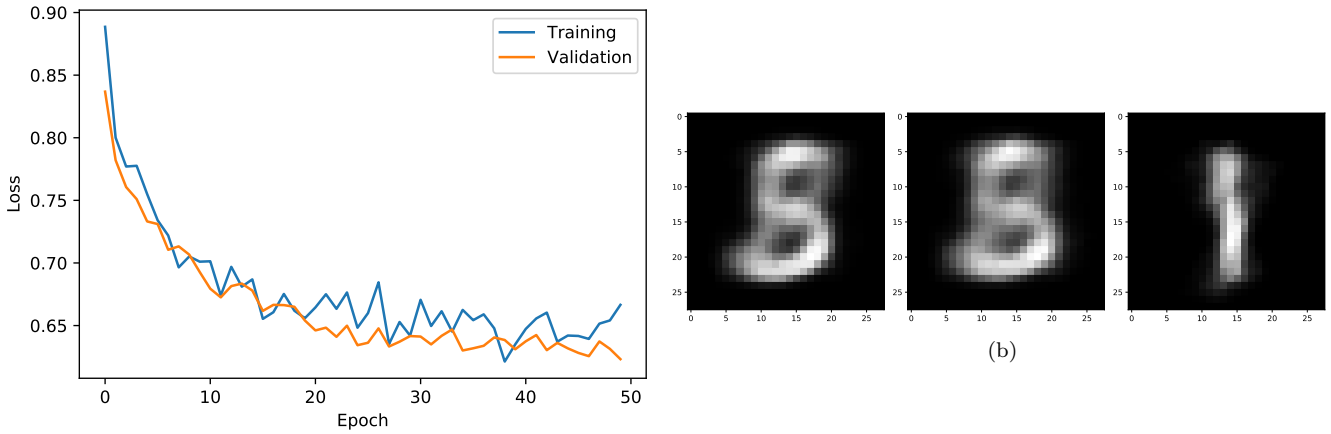


FIG. 5 Example of new samples generated from a random gaussian distribution.



(a)

(b)

FIG. 6 Variational Autoencoders. (a) Training and validation loss profile for training epochs. (b) Example of new samples generated from a random distribution.