UNSUPERVISED DEEP LEARNING

Matteo Bortoletto, 1142935

June 25, 2021

1 Introduction

In unsupervised learning we want to learn patterns from unlabeled data. The hope is that, through mimicry, the machine is forced to build a compact internal representation of its world and then generate imaginative content. In contrast to supervised learning where data is labeled, unsupervised learning exhibits self-organization that captures patterns as neuronal predilections or probability densities.
Unsupervised learning brings with it several advantages. First, given that learning does not require labeled data, we can exploit much more data (in general, most of the data we have in real applications is unlabeled, and if we need labels this has to be done "by hand"). Furthermore, once an unsupervised model has been trained, we can easily use it for supervised tasks, as we will do in this work.
However, unsupervised learning presents also some shortcomings. It is often computationally demanding, and it is hard (almost impossible) to infer causal relationships by passive observation.

1.1 Autoencoders

In this work we focus on autoencoders, which are neural networks able to learn efficient data codings in an unsupervised manner. In other words, the aim of autoencoders is to learn a representation (encoding) for a set of data, typically for dimensionality reduction tasks. This is done by training the network to ignore signal "noise". Along with the reduction side, a reconstructing side is learned, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input (decoding). A schema of a basic autoencoder is shown in Figure 1.
If the number of hidden units is greater than the number of visible ones, then the network will just learn a copy of the input. Conversely, if the number of visible units is greater than the number of hidden ones, the network will (hopefully) extract only the relevant features.
Variants exist, such as:

- *convolutional autoencoders*, which use convolutional layers. This is quite useful when we are dealing with images;
- *denoising autoencoders*, which corrupt the data on purpose by adding noise. This is useful to avoid that the autoencoder learns the so called identity function, not performing representation learning or dimensionality reduction;
- *variational autoencoders*, which makes the mapping from representation space and visible units probabilistic: instead of a point, the encoder returns a distribution over the latent space.

In this work we will train this kind of architectures on the MNIST dataset, which is composed of handwritten digits (from 0 to 9) images.

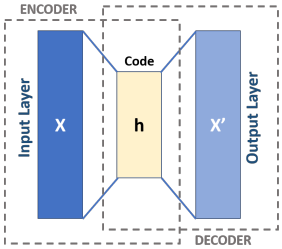


Figure 1: Schema of a basic autoencoder.

2 Convolutional autoencoder

2.1 Methods

The encoder architecture is made of a convolutional section and a linear section. The convolutional section is composed of:

- a first convolutional layer with 8 filters of size 3×3, stride 2 and padding 1;
- a second convolutional layer with 16 filters of size 3×3, stride 2 and padding 1;
- a third convolutional layer with 32 filters of size 3×3, stride 2 and padding 0.

The linear section is composed of:

- a first linear layer with input size 288 and output size 64;
- a final linear layer with input size 64 and output size equal to the encoded space dimension, which is a parameter to be learned.

In between the convolutional section and the linear section the output is flattened. The decoder architecture is symmetric.
To obtain a more powerful mapping, we always use a ReLU activation function, which is non-linear and very efficient. To improve feature extraction, we also use regularization. In particular, we choose the Adaptive Moment Estimator (Adam), a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.
The hyper-parameters we want to optimize are the following:

- the encoded space dimension, which we want to keep relatively small;
- the learning rate;
- the regularization value (weight decay);
- the maximum number of epochs.

Since the dataset is quite large, it is not necessary to use cross validation to avoid overfitting and we find the best value for these parameters through a *grid search*. Furthermore, we use an *early stopping* procedure, stopping the training if the validation loss does not decrease sufficiently in a certain number of consecutive epochs. In this way, if a network does not perform well because it has a poor initialization in terms of hyper-parameters, it will be discarded in a fast way.
As loss function we use the mean squared error (MSE) computed between the original data and the reconstructed image.

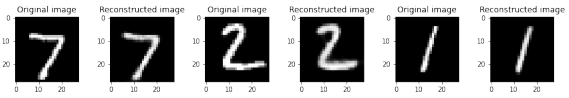


Figure 2: Three examples of reconstruction on test set done by the convolution autoencoder.

Figure 3: Three examples of generated by giving the convolutional autoencoder random gaussian distributed data.

2.2 Results

The grid search leads to the following values for the hyper-parameters:

- encoded space dimension: 16;
- learning rate: 10^{-3};
- weight decay: 10^{-5};
- maximum number of epochs: 150.

Train and validation losses obtained by training the autoencoder with these parameters are shown in Figure 9. Examples of reconstruction for the test set are shown in Figure 2. As we can see, the autoencoder performs the reconstruction pretty well.
If we instead give random gaussian distributed data as input to the autoencoder, we see that the reconstruction is not clear. Figure 3 show some examples. As we can see, the results are not easily interpretable: this is one of the main limitations of "classical" autoencoders, which led to the introduction of variational autoencoders.

3 Denoising autoencoder

3.1 Methods

The denoising autoencoder presents the same architecture discussed in Section 2.1. Also the parameters are the same of the convolutional autoencoder. The only difference is that we feed the encoder with noisy images. In particular, we use a gaussian noise.

3.2 Results

Figure 4 shows some examples of denoising. As we can see, the implementation works quite well – with a test loss of 0.0269 – even if sometimes we see a kind of "uncertainty". In fact, if we look at the first example in Figure 4 we see that the original image is a "7" and also the denoised image is almost a "7", but it is reminiscent of a "9".

4 Fine tuning for classification task

4.1 Methods

If we have a pre-trained encoder we can use *transfer learning* to build a classifier. The idea is to add on top of the encoder some layers and train only them. In particular, we add an intermediate fully-connected layer with 64 units and a final readout layer with 10 output units, equal to the number of classes. The activation between the two layers is a ReLU and after the readout layer we use a log-softmax, which improves efficiency. As loss function we use the negative log-likelihood, which performs well if we are dealing with log-probabilities.

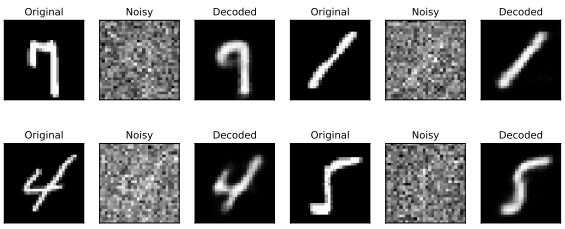


Figure 4: Examples of original image, noisy transformation and reconstruction by the denoising autoencoder.

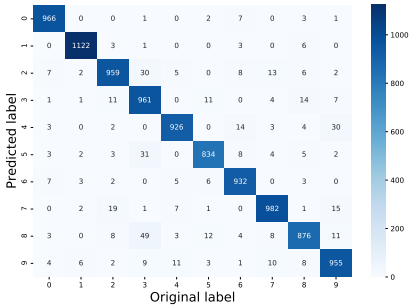


Figure 5: Confusion matrix obtained for the fine-tuned classifier based on the pre-trained convolutional autoencoder.

4.2 Results

The confusion matrix obtained on the test dataset is shown in Figure 5. As we can see, most of the labels given by the model are correct. Notice that the highest number of misclassification errors (49) is made by mistaking a "3" for a "8", which is understandable since the two digits are very similar. Other cases of frequent misclassification are "9" and "4", "3" and "2", and "3" and "5".
The test accuracy is 0.9513, which is slightly lower than the one obtained with the convolutional neural network in the previous work, which was 0.9927. Nevertheless, such high accuracy means that transfer learning is a powerful technique.

5 Variational autoencoder

5.1 Methods

In standard autoencoders, the latent space can be extremely irregular, i.e. close points can produce very different patterns. A possible fix is to make the mapping between latent space and visible space probabilistic. In this case the encoder does not return a point but a probability distribution over the latent space, and the loss function has an additional regularization term to ensure a "better organization" of the latent space.
The encoded distribution is often chosen to be a multivariate gaussian, so that the encoder outputs a vector with means $\boldsymbol{\mu}$ and covariances $\boldsymbol{\sigma}$ of the input distribution. Moreover, to reduce computational complexity we assume the covariance matrix to be diagonal, meaning that latent variables are independent (as in RBMs).
The loss function is regularized by forcing the latent distribution to be as close as possible to the Normal distribution:

$$\text{loss} = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 + \text{KL}(\mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x)||\mathcal{N}(\mathbf{0}, \mathbf{1})) \tag{1}$$

where $\boldsymbol{x}$ is the original input, $\hat{\boldsymbol{x}} = \text{decoder}(\boldsymbol{z})$ is the reconstructed input and $\text{KL}(P||Q)$ denotes the Kullback–Leibler divergence, which is a measure of how one probability distribution P is different from a second,

Figure 6: Convolutional variational autoencoder samples generation.

reference probability distribution Q.
The only problem with this approach is that sampling is a discrete process, so we cannot use backpropagation. However, we can solve it by reparametrizing the sampled latent representation $\boldsymbol{z}$ as

$$\boldsymbol{z} = \boldsymbol{\sigma}_x \boldsymbol{\zeta} + \boldsymbol{\mu}_x, \qquad \text{with } \boldsymbol{\zeta} \sim \mathcal{N}(\mathbf{0}, \mathbf{1}), \tag{2}$$

and then backpropagate.
The architecture we use for the encoder is the following:

- a first convolutional layer with 64 filters of size 4×4, stride 2 and padding 1;
- a second convolutional layer with 128 filters of size 4×4, stride 2 and padding 1;
- two linear layers, one for the means and one for the log-covariances.

For the decoder, we have:

- a linear layer with number of inputs equal to the latent dimension;
- a first de-convolutional layer with 128 filters of size 4×4, stride 2 and padding 1;
- a second de-convolutional layer with 64 filters of size 4×4, stride 2 and padding 1;

5.2 Results

As we have seen in Section 2.2, if we give random gaussian distributed data as input to the classical autoencoder we see that the reconstruction is not clear (Figure 3). On the contrary, if we look at Figure 6, the variational autoencoder seems to perform better. However, we can still see that sometimes the reconstruction is not perfectly clear. For example, it is not easy to understand if the last digit in the second row is a "9" or a "4". Another example is the second digit in the second row, which can be both a "9" or a stretched "8".

6 Latent space analysis

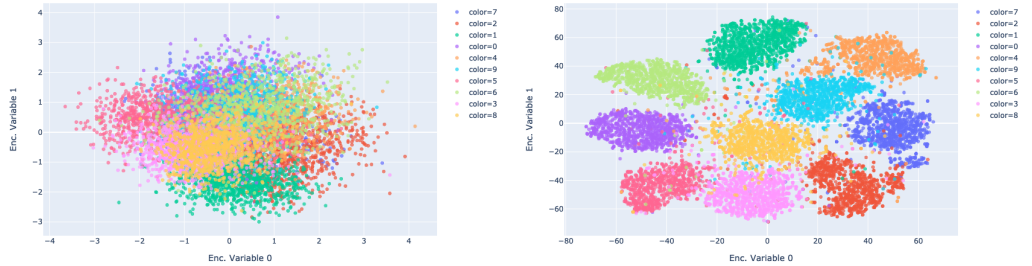
6.1 Methods

In this section we explore the latent space structure. To do that we use two dimensionality reduction tools:

- the *Principal Component Analysis* (PCA), which is the process of computing the principal components of the data – i.e. the dimensions with higher variance – and using them to perform a change of basis on the data, projecting each data point onto only the first few principal components. In this way we obtain lower-dimensional data while preserving as much of the data variation as possible;

(a) PCA. (b) t-SNE.

Figure 7: PCA and t-SNE results for the convolutional autoencoder.



(a) PCA. (b) t-SNE.

Figure 8: PCA and t-SNE results for the convolutional variational autoencoder.

- the *t-distributed stochastic neighbor embedding* (t-SNE). First, it constructs a probability distribution over pairs of high-dimensional objects in such a way that similar objects are assigned a higher probability while dissimilar points are assigned a lower probability. Second, it defines a similar probability distribution over the points in a low-dimensional map, and it minimizes the Kullback-Leibler divergence between the two distributions with respect to the locations of the points in the map. In other words, t-SNE models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability.

6.2 Results

The results obtained for the classical convolutional autoencoder are shown in Figure 7. As we can see from Figure 7a, the PCA result is not very good, as the various digits clusters tend to overlap. This is more evident for digits that are quite similar, for example "7" (blue points) and "9" (cyan points). Conversely, from Figure 7b we can see that t-SNE produces good results, that is clusters are well separated.
The results obtained for the variational convolutional autoencoder are shown in Figure 8. Both Figure 8a and Figure 8b show that the cluster are better separated in this case. This means that variational autoencoders have a more regular latent space. As for the classical convolutional autoencoder, t-SNE works better.

A Appendix

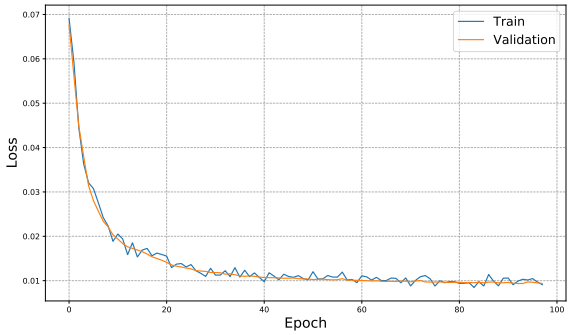


Figure 9: Train and validation losses obtained by training the convolutional autoencoder with the best hyper-parameters.