Machine Learning for Signal Processing

[5LSL0]

Ruud van Sloun
Rik Vullings

Assignments (Variational) Autoencoders

May 16, 2019

Autoencoders

In this assignment you will implement and analyze and autoencoder and study the difference between deterministic and variational autoencoders.

For question where you need to code in Python, provide the most relevant lines of code and show the plots generated with the code in your report.

Deterministic autoencoder

An autoencoder is a neural network that attempts to copy its input to its output via latent space $\mathbf{h}$. The network can be considered to consist of two parts: an encoder and a decoder. The encoder maps the input $\mathbf{x}$ to the latent space $\mathbf{h} = f(\mathbf{x})$. The decoder aims to reconstruct the inputs from the latent space $\mathbf{r} = g(\mathbf{h})$. The autoencoder is successful when $g(f(\mathbf{x})) = \mathbf{x}$. The value of a perfect autoencoder is however limited.

Typically, we aim to design the autoencoder in such a way that it is unable to perfectly reconstruct the inputs. That way, the autoencoder is forced to copy only input that resembles the training data, prioritizing useful properties of the data. Applications of autoencoders therefore include:

- Dimensionality reduction
- Feature learning
- Denoising

In this assignment you will need to create an autoencoder that operates on the idea that data concentrates around a low-dimensional manifold. The aim of autoencoders is to learn the structure of the manifold. More specifically, in this assignment you will use the MNIST database, a large public database of handwritten digits (0-9) that is commonly used for training various image processing systems.

As a first step you will need to install TensorFlow and Keras on your systems so that you can exploit these powerful libraries for designing, implementing, and training your autoencoder.

Exercise 1

Create a convolutional autoencoder that loads batches of data from the MNIST dataset. You can use Keras libraries and a sample code is already available for you to complete. Requirements for the network are as follows:

- You can only use Convolutional layers (Conv2D), Pooling, and Upsampling.
- After every Convolutional layer, you must use rectified linear units (ReLU) as activation.
- Use padding='same' for the Convolutions
- After every Convolutional (+activation) layer there must be a Pooling or Upsampling layer that decreases or increases, respectively, the dimension of the network by a factor 2
- Convolutional layers must all use 16 different filters with a kernal size of $(3, 3)$. Exceptions are the last layers of both the Encoder and Decoder (see below).
- Use to different functions, one for the Encoder and one for the Decoder
- The last steps in the Encoder should be designed such that the output of the Encoder has dimensions $[Nx1x2x1]$, where N is the batch size. In other words, for each image, the latent space has dimensions $[1x2x1]$, the last 1 meaning that the last Convolutional layer in your Encoder should have only 1 filter and your last Pooling step should be designed such that it produces a $[2x1]$ array/tensor.

Visualize the first 10 images of `x_batches` next to the model.predict of this batch. What differences/resemblances you do notice between the input and output of the autoencoder? Also make a plot of the loss as a function of the number of iterations or epochs.

Exercise 2

(a) Create a Python script that visualizes the latent space that is provided by the Encoder for all the MNIST images. To this end, determine the output of the Encoder and make a scatter plot where you use the $[2x1]$ vector of the latent space as values for the horizontal and vertical axes in a 2D scatter plot. Give each datapoint in the scatter plot a color that corresponds to the digit that is written in the original image (use `data_y`).

(b) Which digits provide clear clusters and which overlap? Can you think of an explanation for this?

(c) The datapoints, especially for some of the clusters, are clipped at weight of zero. Explain why this has happened.

Exercise 3

Although our network was not trained for classification, we can use the latent space to do some rudimentary classification of the handwritten digits in their correct class, classes being defined as the digits that are written in the images (i.e. `data_y`).

(a) Encode the test data `data_x_test` and perform a 1-nearest neighbour search. You can use the sci-kit learn library for nearest neighbour implementations:
`from sklearn.neighbors import NearestNeighbors`
In the 1-nearest neighbour, for every datapoint in the latent space of the testset, you search the datapoint from the training set that has the smallest Euclidean distance in the latent space. You can consider this such that the two images are mapped to the same area in the latent space and hence have relatively high probability that their characteristics, hopefully the digits they represent, are similar. Therefore, assign the

class to the test image that corresponds with the class `data_y_train` of its nearest neighbour.

For the 1000 images in the testset, determine how many zeros, ones, twos, etc. were classified correctly and express these 10 accuracies as percentages of the total number of zeros, ones, twos, etc.

(b) If the classification based on the latent space would not work at all and produce random datapoint in the latent space, how many correct classifications would you expect?

(c) Which numbers are classified significantly better than the other, and (consequently) which number aren't? Can you explain this from the plot of the latent space you made in Exercise 2?

(d) If we would design our network differently, where our goal would not be to encode and decode the images but to correctly classify the images, what cost function would you use?

Exercise 4

As mentioned above, the autoencoder is not designed for classification, but still it manages to classify at least some of the digits relatively accurate. With a rather simple modification, it should be possible to convert your network into a network that is designed for classification purposes.

(a) Use your Encoder as starting point and replace the last Convolutional layer (and its associated Activation and Pooling layer) and replace these by a Fully-Connected, or its equivalent: Dense, layer. If necessary, you can use a Flatten layer as well. Assign a proper activation function and train and test your network on exactly the same training and test data.

What accuracies on the test set can you achieve now?

(b) Provide graphs of the training and test loss (perform intermittent evaluations of the test loss during training). Comment on the capacity of your network.

(c) Give some suggestions on improvements in case you were underfitting or in case you were overfitting.

Variational autoencoder

Variational autoencoders are a special type of autoencoders with added constraints on the encoded representations that are being learned. More precisely, they are autoencoders that learn a latent variable model for the input data. So the autoencoder learns the parameters of a probability distribution that models you data. If you would sample points from this distribution, you could use the Decoder stage of the autoencoder to generate new samples of the input data. As such, you can consider variational autoencoders as so-called generative models.

Exercise 5

In this exercise you are not going to develop a variational autoencoder (although you are encouraged to try this as well) but we are going to use our previously trained autoencoder – in particular the Decoder stage – as a generative model.

(a) Define a 15 x 15 grid that equidistantly samples the latent space between 0 and 0.6 across both axes. In other words, if you visualize the latent space as a Cartesian coordinate system, sample between the points (0,0), (0,0.6),(0.6,0),(0.6,0.6) with 15 steps in horizontal and vertical directions.

Use these samples from the latent space as inputs to your Decoder to create new images. Plot all images (`pyplot.imshow`) in one figure where you define the axes of the figure such that the grid coincides with the grid of the scatter plot in Exercise 2

(b) Which digits can you recognize? how do the images change from left to right and top to bottom? Can you explain this (use the scatter plot of Exercise 2 to do so)?

(c) Explain the differences between the generative model you used here and a generative model based on a variational autoencoder.