Autoencoders and Variational Autoencoders

MVA, Cours Delires

February 28, 2019

The goal of this TP is to explore autoencoders and variational autoencoders applied to a simple dataset. Your job is to fill in the parts indicated with "**FILL IN CODE**" and to answer the questions in this document. If you are using the local machines, in order to use the Keras package, you must first load an anaconda environment with these packages. To do this, execute the following commands in a terminal :

```
mv .conda .conda_bak # If you already have a conda installation, otherwise, ignore this
ln -s ~anewson/.conda .conda
export PATH=/cal/softs/anaconda/anaconda3/bin:$PATH # Enable anaconda
source activate keras_env #activate the environment with keras etc
```

If you have your own machine, you can install you own environment. You will need the following packages :

- keras
- matplotlib
- numpy

We will be using the "mnist" dataset. This is a set of 60,000 greyscale images, of size 28×28. The images are of handwritten digits. This simple dataset is easy to manipulate and experiment with, since the images are not too large, so training does not take too long. Feel free to try this code on other datasets to see how they work.

1 Autoencoders

In the first part of the TP we will implement a simple autoencoder (AE), with two different architectures :

- Multi-layer perceptron (fully-connected layers)
- Convolutional network

We are going to implement our architectures with the Keras package (these are already loaded in the python code tp_ae.py). Here is an example of the definition of a simple one-layer neural network :

```
x = Input(shape=(rows,cols,channels))
x_flatten = Flatten()(x)
z = Dense(z_dim)(x_flatten)
y = Activation('sigmoid')(z)
mlp = Model(x,y)
```

You can also carry out both the fully-connected layer and non-linearity in one go with :

```
x = Input(shape=(rows,cols,channels))
x_flatten = Flatten()(x)
z = Dense(z_dim, activation='sigmoid')(x)
mlp = Model(x,z)
```

For the definitions of the necessary architectures, you can use the following Keras layers and operations :

- Flatten
- Dense
- Activation
- LeakyReLU
- Reshape
- Conv2D
- Conv2DTranspose

Please look at the online Keras documentation[1] for more information about these functions. There are other APIs which exist in Keras (such as the Sequential() API, you may use these if you like).

1.1 Notation

Let $x \in \mathbb{R}^{mn}$ be the input to the AE, and $y \in \mathbb{R}^{mn}$ be the output. Furthermore, let $z \in$ be the *latent code*, in other words z is the projection of x onto the latent space. This projection is done with the encoder : $\Phi_e : \mathbb{R}^{mn} \rightarrow \mathbb{R}^d$. Let the decoder be $\Phi_d : \mathbb{R}^d \rightarrow \mathbb{R}^{mn}$.

1.2 Autoencoder architecture

We are going to train the AE on the mnist dataset. The loading and pre-processing of the dataset is already coded in tp_ae.py. Your task is to implement the following two different architectures for the encoder and decoder :

- MLP :
  - Encoder :
    1. Flatten input x, to size 784
    2. Dense layer, output size d
    3. Leaky ReLU ($\alpha = 0.2$)
  - Decoder :
    1. Dense layer, output size 784
    2. Sigmoid activation
    3. Reshape, to size $28 \times 28 \times 1$
- Convolutional network :
  - Encoder :
    1. 2D Convolutional layer (with bias), filter size : 3×3, 8 filters, stride = $(2, 2)$, padding : same
    2. Leaky ReLU ($\alpha = 0.2$)
    3. 2D Convolutional layer (with bias), filter size : 3×3, 4 filters, stride = $(2, 2)$, padding : same
    4. Leaky ReLU ($\alpha = 0.2$)
    5. Flatten tensor
    6. Dense layer, output size d
  - Decoder :

[1] `https://keras.io/`

1. Reshape ($7 \times 7 \times 8$)
2. 2D transposed convolution, filter size : 3×3, 4 filters, stride = $(2, 2)$, padding : same
3. Leaky ReLU ($\alpha = 0.2$)
4. 2D transposed convolution, filter size : 3×3, 8 filters, stride = $(2, 2)$, padding : same
5. Sigmoid activation

Questions:

1. Implement the MLP architecture, with $d = 10$
2. Implement the convolutional architecture, with $d = 10$
3. Compare the results of the two architectures, in terms of the loss attained and in terms of visual results. Which one leads to better results and/or leads to a lower loss ? Why do you think this is ?
4. Modify the code to create a *denoising autoencoder*. You can use a standard deviation of the noise of $\sigma = \frac{20}{255}$, for example. Do the results look satisfactory to you ?

2 Variational autoencoder

We are now going to implement a Variational Autoencoder (VAE), with the goal of image synthesis. We are going to consider the VAE in the case of a Gaussian encoder and a Benoulli decoder. This situation is ideal for modelling the "mnist" dataset. More precisely, we have the following configuration :

- The approximation of the posterior is Gaussian : $q_\phi(z|x) \sim \mathcal{N}(\mu, \sigma^2)$, where $\mu \in \mathbb{R}^d$ is the mean vector, an output of the encoder, and σ^2 is a diagonal covariance matrix, another output of the encoder
- The prior is Gaussian : $p_\theta(z|x) \sim \mathcal{N}(0, Id)$
- The likelihood is a Bernoulli : $p_\theta(x|z) \sim Ber(y)$ for each pixel, where y is the output of the decoder

2.1 The VAE loss function

The VAE loss consists of the sum of two losses : the reconstruction loss, and the KL divergence between the posterior and the prior distribution. We wish to *maximise* the following loss :

$$\mathcal{L} = \mathbb{E}_{q_\phi}\left[\log p_\theta(x|z)\right] - KL(q_\phi(z|x) \;||\; p_\theta(z)). \tag{1}$$

To implement this loss in Keras, it is more practical to *minimise* $-\mathcal{L}$. These losses depend on the probability distributions which we have chosen to model the autoencoding problem.

2.1.1 Encoding : the Gaussian case

For two Gaussian distributions $P_1 = \mathcal{N}(\mu_1,\ \Sigma_1)$ and $P_2 = \mathcal{N}(\mu_2,\ \Sigma_2)$, where $\mu_1 \in \mathbb{R}^d$, $\Sigma_1 \in \mathbb{R}^{d \times d}$ and similarly for P_2, the KL divergence can be calculated as :

$$KL(P_1 \;||\; P_2) = \frac{1}{2}\left(\text{trace}(\Sigma_2^{-1}\Sigma_1) + (\mu_2 - \mu_1)^T \Sigma_2^{-1}(\mu_2 - \mu_1) - d + \log \frac{\det \Sigma_2}{\det \Sigma_1}\right). \tag{2}$$

Question :

Determine the KL divergence loss in our situation, that is to say where $P_1 = \mathcal{N}(\mu,\ \Sigma)$ is the distribution of the approximate posterior $q_\phi(z|x)$, with Σ a diagonal matrix, and $P_2 = \mathcal{N}(0,\ Id)$ is the prior.

2.1.2 Decoding : the Bernoulli case

We consider that the decoding process $p_\theta(x|z)$ follows a Bernoulli distribution, that is to say each pixel is a Bernoulli random variable which follows a parameter $y = \Phi_d(z)$, which is the decoded output of the VAE (y is an image of Bernoulli parameters).

Question :

Given the observed input x, calculate the reconstruction loss $\log p_\theta(x|z)$ in this case. Important note : although this is not strictly true in practice, we make the following simplifications :

- The output pixels in the image are independent random variables
- The observation x is a binary image

2.2 Implementing the VAE

We are going to use the following MLP-type architecture :

- Encoder :
  1. Flatten input x, to size 784
  2. Dense layer, output size 512
  3. Leaky ReLU ($\alpha = 0.2$)
  4. z_mean $\leftarrow$ Dense layer (z) (to output size d)
  5. z_log_var $\leftarrow$ Dense layer (z) (to output size d)
- Decoder :
  1. Sample the random vector z (with z_mean, z_log_var, this is already implemented)
  2. Dense layer, output size 512
  3. Leaky ReLU ($\alpha = 0.2$)
  4. Dense layer, output size 784
  5. Sigmoid activation

Questions

1. Implement the VAE architecture, with $d = 10$. **Important note** : the "binary_crossentropy" loss function in Keras calculates the *mean* of the individual cross-entropies of the pixels[2]. If you decide to use this function, either multiply it by the number of pixels to scale it, or you can use the backend binary cross-entropy ("K.binary_crossentropy") and then take the sum over the last axis (the second option is more rigourous).
2. Fill the function "sample_images" to create a function which randomly samples from a normal distribution and outputs the synthesised images.
3. If you have time left over, change the architecture of the network to be a convolutional architecture.

[2]See `https://github.com/keras-team/keras/blob/master/keras/losses.py`