Exercise Sheet 6: Autoencoder

Due on 07.06.2024, 10:00

Olga Grebenkova (o.grebenkova@lmu.de)

Important Notes:

1. **Questions:** Please use the Moodle platform and post your questions to the forum. They will be answered by us or your fellow students.
2. **Submission:** Your submission should consist of one single ZIP file which includes a PDF file and the corresponding codes. Both the PDF file and ZIP file should contain your surname and your matriculation number (Surname-MatriculationNumber.zip) for grading purposes. You may use Jupyter 1 for exporting your python notebooks as PDF, but you still have to hand in your .ipynb or .py files for us to test your code. For this exercise, please export the .ipynb as a PDF file and include that in the ZIP file. Submissions that fail to follow the naming convention or missed PDF/code files will not be graded.
3. **Deadline:** The due date for this exercise is the 7th of June.

Task 1: Implement a Linear Autoencoder (10P)

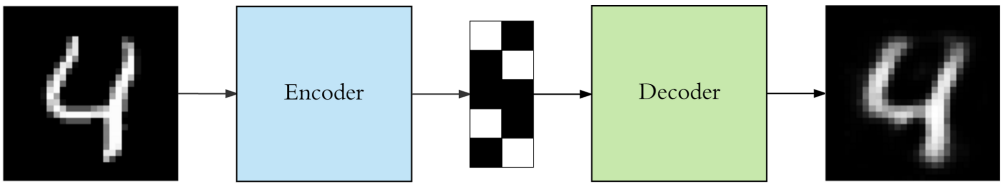

Figure 1: The classical autoencoder architecture consists of an encoder and a decoder. An autoencoder takes an high-dimensional input and attempts to reconstruct it despite passing it through an "information bottleneck", a low-dimensional code.

We consider the autoencoder problem (see Fig. 1) that aims at reconstructing an input x that is passed through a lower-dimensional bottleneck h. The learning problem is described as minimizing a loss function

$$L(x, g(f(x))), \tag{1}$$

where x is the input, $f(x)$ the encoder that produces the code h, and $g(x)$ the decoder. L is a loss function penalizing $g(f(x))$, the output of the autoencoder, for being dissimilar from x, such as the mean squared error (MSE).

In the following task you will implement an autoencoder as a neural network to solve (1). Base your code on PyTorch and make use of the fashionMNIST dataset for training and testing. Fashion-MNIST[1] contains images of 10 different types of clothes and shoes (e.g., pullovers, dresses, sneakers) but in low-resolution. It has exactly the same amount of data as classical MNIST and the same number of classes. That means you can easy replace one by another without changing too much.

1. Implement a fully-connected autoencoder using the class torch.nn.Module[2] from pytorch. Use the provided starter code as a template. The architecture of encoder and decoder should look as follows:

   - Encoder
     1. Linear layer with 128 neurons followed by ReLU
     2. Linear layer with 64 neurons followed by ReLU
     3. Linear layer with 32 neurons followed by ReLU
     4. Linear layer with 16 neurons followed by ReLU
     5. Linear layer with 8 neurons followed by ReLU

   - Decoder
     1. Linear layer with 8 neurons followed by ReLU
     2. Linear layer with 32 neurons followed by ReLU
     3. Linear layer with 64 neurons followed by ReLU
     4. Linear layer with 128 neurons followed by ReLU
     5. Linear layer with 784 neurons followed by Tanh

2. Use the mean squared error[3] to implement the reconstruction error of the autoencoder. The error will be computed between the input image x and the output of the autoencoder $g(f(x))$.

3. For training use Adam as optimizer with a learning rate of 0.001, weight decay of $1e-5$ and a batch size of 128. Run it for at least 50 epochs. Report

[1] https://github.com/zalandoresearch/fashion-mnist
[2] https://pytorch.org/docs/master/generated/torch.nn.Module.html
[3] https://pytorch.org/docs/master/generated/torch.nn.MSELoss.html

the loss on the training split for each epoch, and every ten epochs on the test set.

4. During the computation of the testing loss plot ten input samples as well as their reconstructions produced by the autoencoder. You should do it every 10 epochs.

Task 2: Denoising autoencoder (3P)

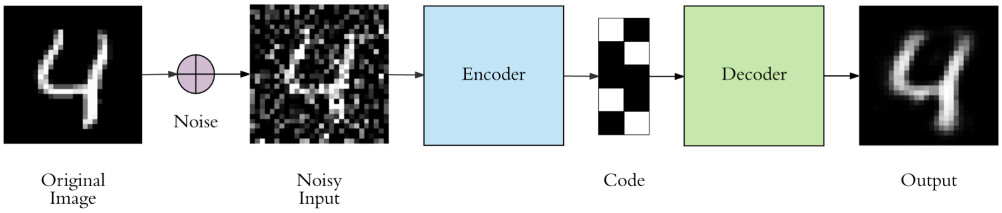

Figure 2: Denoising autoencoders are a specific type of neural network that enables unsupervised learning of data representations or encodings. Their primary objective is to reconstruct the original version of the input signal corrupted by noise. This capability proves valuable in problems such as image recognition or fraud detection, where the goal is to recover the original signal from its noisy form.

It is also possible to learn Denoising Autoencoder (see Fig. 2). It is a neural network that remove noise from corrupted or noisy data by learning to reconstruct the initial data from its noisy counterpart. Your task is to train the model to minimize the disparity between the original and reconstructed data.

- Use your code for the previous task. The model stays the same. But instead of the original input apply your network to the noised input (see function *add noise* in *network.py*).
- Note that you should still count loss between not noised image and your prediction.
- Describe how has the performance of the model changed.

Task 3: Implement a Convolutional Autoencoder (7P)

In the following task you will implement an autoencoder as a neural network to solve (1), but with different architecture. Base your code again on PyTorch and make use of the fashionMNIST dataset for training and testing.

1. Implement a convolutional autoencoder using the class torch.nn.Module[4] from pytorch. Use the provided starter code as a template. The architecture of encoder and decoder should look as follows:s

   - Encoder
     1. Convolutional layer with 1 input channel, 4 output channels and kernel size=5 followed by ReLU
     2. Convolutional layer with 4 input channels, 8 output channels kernel size=5 followed by ReLU
     3. nn.Flatten (to flatten our features)
     4. Linear layer with 10 neurons
     5. Softmax module

   - Decoder
     1. Linear layer with 400 neurons followed by ReLU
     2. Linear layer with 4000 neurons followed by ReLU
     3. nn.Unflatten (to return channels)
     4. Transposed convolutional layer with 10 input channels, 10 output channels, kernel size=5 followed by ReLU
     5. Transposed convolutional layer with 10 input channels, 1 output channel, kernel size=5 followed by Tanh

2. Use the mean squared error[5] to implement the reconstruction error of the autoencoder. The error will be computed between the input image x and the output of the autoencoder $g(f(x))$.

3. For training use Adam as optimizer with a learning rate of 0.001, weight decay of $1e-5$ and a batch size of 128. Run it for at least 50 epochs. Report the loss on the training split for each epoch, and every ten epochs on the test set.

4. During the computation of the testing loss plot ten input samples as well as their reconstructions produced by the autoencoder. You should do it every 10 epochs.

[4] https://pytorch.org/docs/master/generated/torch.nn.Module.html
[5] https://pytorch.org/docs/master/generated/torch.nn.MSELoss.html

5. Compare in 2-3 sentences the performance of this model to the model from the first task.