UNIVERSITÄT HEIDELBERG
Zukunft. Seit 1386.
IWR Interdisciplinary Center for Scientific Computing

Artificial Intelligence
Wintersemester 17/18
Prof. Björn Ommer
ommer@uni-heidelberg.de

Exercise sheet 9: SGD and Autoencoders

Due on *12/01/2018*, 2pm.

Question 1: Variants of SGD **(3P)**

Please describe in your words the following variants of gradient descent: *stochastic* gradient descent, *batch* gradient descent, and *mini-batch* gradient descent. For each variant state at least one advantage and disadvantage.

Question 2: Implementing an Autoencoder **(4P)**

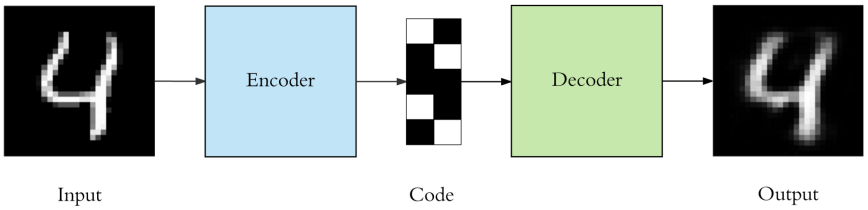

Figure 1: The classical autoencoder architecture consists of an encoder and a decoder. An autoencoder takes an high-dimensional input and attempts to reconstruct it despite passing it through an "information bottleneck", a low-dimensional code.

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 (introduced in exercise 8) for training and testing.

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

[1] `http://pytorch.org/docs/master/nn.html#torch.nn.Modules`

- 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

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

c) For training use *Adam* as optimizer with a *learning rate* of 0.001, *momentum* 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.

d) During the computation of the testing loss plot ten input samples as well as their reconstructions produced by the autoencoder.

One useful property of the autoencoder is that it learns to compress a high-dimensional input like an image into a low-dimensional code. Compression is often achieved by leveraging the common structure in the dataset. This *dimensionality reduction* does not require expensive annotations (unsupervised learning) and can provide useful insights into the data.

e) After having trained the autoencoder compute for 500 images of the test set their respective code representation $f(x)$. Apply the t-SNE visualization technique (introduced in exercise sheet 8) to obtain 2D coordinates and plot the result as a scatter plot. Use a discrete colormap to visualize the different class labels and give a small description how well the code is organizing the test images in the 2D plane.

[2] `http://pytorch.org/docs/master/nn.html#torch.nn.MSELoss`

Question 3: Using an Autoencoder for Image Denoising **(3P)**

Besides learning a low-dimensional representation of the data, autoencoders can also be used for the task of image denoising. We consider again the fashionMNIST dataset as shown in Fig. 2.

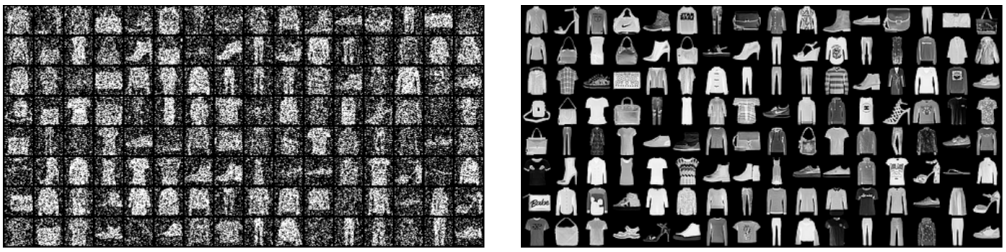

Figure 2: Data samples from the fashionMNIST dataset with added white noise. The goal in image denoising is to reconstruct the original image from the noisy version.

a) In the code provided with the assignment you find a function which adds white noise (normal distributed noise) to a tensor. Modify your autoencoder training code from the previous question to obtain a denoising autoencoder as shown in Fig. 3. It should take a noisy image as input while still using the original version for computing the reconstruction loss at the output.

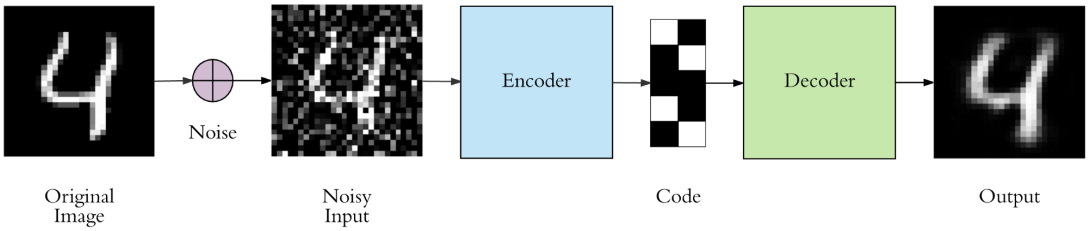

Figure 3: The denoising autoencoder takes a noisy input and reconstructs the original by passing it through an encoder-decoder architecture.

b) Train the denoising autoencoder with the training setup used for the normal autoencoder. Similar to previous question plot the noisy input, the reconstruction as well as the original for ten test samples every 10 epochs.

c) Finally, use t-SNE to analyze the code representation obtained from the training of the denoising autoencoder. Is it different from the code learned by the normal autoencoder?

Note: Submit exactly one ZIP file and one PDF file via Moodle before the deadline. The ZIP file should contain your executable code. Make sure that it runs on different operating systems and use relative paths. Non-trivial sections of your code should be explained with short comments, and variables should have self-explanatory names. The PDF file should contain your written code, all figures, explanations and answers to questions. Make sure that plots have informative axis labels, legends and captions.