Exercise 2

Deadline: 22.11.2023 16:00.
Ask questions in discord to #ask-your-tutor

In this exercise, you will implement and test autoencoders.

Regulations

Please implement your solutions in form of *Jupyter notebooks* (`*.ipynb` files), which can mix executable code, figures and text in a single file. They can be created, edited, and executed in the web browser, in the stand-alone app JupyterLab, or in the cloud via Google Colab and similar services.

Create a Jupyter notebook `autoencoders.ipynb` for your solution and export the notebook to HTML as `autoencoders.html`. Zip all files into a single archive `ex02.zip` and upload this file to MaMPF before the given deadline.
Moreover, please set your **Anzeigename/display name** and **Name in Uebungsgruppen/name in tutorials** in MaMPF to your real name, which should be identical to your name in `muesli` and make sure you **join the submission** of your team via the invitation code before the submission deadline. Check out https://mampf.blog/handing-in-homework-assignments for instructions.

1 Two-dimensional data with bottleneck

We again work with `sklearn.datasets.make_moons()` to create 2-dimensional data sets of varying sizes. Use pytorch to implement and train an autoencoder. The autoencoder's constructor should have four arguments:

input_size: the data dimension (so that we can later reuse the autoencoder in task 3)

bottleneck_size: the code dimension

hidden_size: the dimension of the hidden layers (should be larger than the data dimension!)

layers: the number of hidden layers

In the present task, we obviously have `input_size=2` and `bottleneck_size=1`, the other two arguments are hyperparameters. All interior layers should be linear followed by a ReLU. The output layers of both encoder and decoder are just linear. The decoder architecture should be a mirror of the encoder architecture.

Implement a training function `train_autoencoder()` using the `torch.optim.Adam` optimizer (with default learning rate `lr=0.001`) and the mean-squared reconstruction error. Training has thus three hyperparameters: the size of the training set, the number of epochs, and the learning rate (or learning rate schedule). During design and debugging of your code, your should start small (e.g. 1000 training points, 10 epochs, 2 layers) to ensure rapid training. Gradually increase network size and training effort once your results start looking promising.

When your code works, train and evaluate autoencodes with two moons at noise level 0.1.

1. Investigate systematically the effect of the hyperparameters. Report your findings in suitable plots (e.g. test set reconstruction error as a function of hyperparameters, and reconstructed vs. original points for selected settings). Make sure that all plots are properly labeled (title, axis labels, legend). Comment on your findings.
2. For the best hyperparameter settings you found, investigate how much the reconstruction varies when you repeat training with the same or different datasets. Comment on your observations.

3. Create and visualize a histogram of the code distribution. Fit a Gaussian mixture model to the code distribution and use it to sample synthetic data. Comment on the quality of the generated data.
4. Check if the autoencoder still works (without retraining!) on a test set at noise level 0.2.
5. Train an autoencoder with a training set at noise level 0.2 and comment on how the geometry of the reconstructed set changes.

2 Two-dimensional data without bottleneck

We continue with the two moons dataset, but this time bottleneck size 2. Since training now lacks the regularizing effect of the lossy compression (which enforces the codes to focus on the important data features), we need an additional loss term. We use the maximum mean discrepancy (MMD) between the code distribution (i.e. the push-forward through the encoder) and a 2-dimensional standard normal. This loss pulls the code distribution towards the standard normal, so that we can later generate synthetic data by sampling from the standard normal and transform the samples through the decoder. The kernel for MMD shall be a sum of squared exponentials or inverse multi-quadratics at multiple bandwidths (the number and values of the bandwidths are hyperparameters – use three to seven bandwidths such that each value is twice the next smaller one). The training loss is a weighted sum of the squared reconstruction error and the MMD, with the weight being yet another hyperparameter.

Use the autoencoder from task 1 (with `bottleneck_size=2`) or implement a residual network (ResNet – it gave better results in our experiments). Recall that a ResNet consists of L residual blocks computing $z_l = z_{l-1} + f_l(z_{l-1})$ for $l = 1, ..., L$, where z_0 are the original data. The $f_l()$ are fully connected networks with one or two hidden layers whose width is typically larger than the input dimension. Implement a new function `train_mmd_autoencoder()` and train the model at noise level 0.1.

Visualize the reconstructed vs. original data, and the code distribution. Check that the reconstruction error is much less than what you got with a bottleneck of size 1. Generate synthetic data by passing standard normal samples through the decoder and visualize their quality. Comment on your findings.

3 Higher-dimensional data

Repeat task 1 with the digits dataset (`sklearn.datasets.load_digits()`). Test bottleneck sizes 2, 4 and 8. Visualize and measure the quality of the reconstructions as a function of bottleneck size. Can you still recognize the digit labels in the reonstructed data? Train a `sklearn.ensemble.RandomForestClassifier` on the training data and use it to classify the test set and the reconstructed test set. Check if the predicted labels are the same for originals and reconstructions.

For bottleneck size 2, visualize the code distribution and color the points according to their labels. Do you observe interesting structure in the colored code distribution? Fit a Gaussian mixture model to the code distributions for all bottleneck sizes and generate synthetic digits. Visually inspect the quality of the synthetic data – can you recognize their labels? Check the performance of the classifier on the synthetic data – does it agree with what you see?