TECHNISCHE UNIVERSITÄT DRESDEN

Professur für Informationstheorie und maschinelles Lernen, Fakultät Elektrotechnik und Informationstechnik

Prof. Rafael Schaefer, Dr. Martin Mittelbach, Muah Kim, Dr. Rick Fritschek, Max Wolf 26.07.2024

3. Homework of the Course
Einführung in das maschinelle Lernen

For this homework assignment, you have the choice of working on and submitting ***either Problem 4 or Problem 5****. Only one of the two problems is mandatory.*

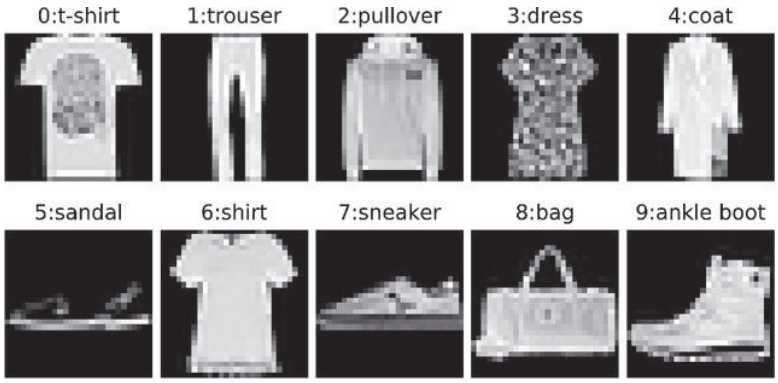
The homework can be submitted by ***12.09.2024, 23:59****.*

Please upload your results via the OPAL repository of this course using the available upload folder named "3. Hausaufgabe".

Upload your results as a single ZIP-file with the name firstname-lastname-homework-03.zip.

Homework Problem 4: (CNN autoencoder for Fashion-MNIST dataset)

The goal is to implement an autoencoder (AE) based on a convolutional neural network (CNN) architecture using PyTorch to learn a compressed representation of the Fashion-MNIST dataset.



Autoencoder architecture:
The considered CNN-based AE generally has an architecture meeting the following specifications:

- The encoder part consists of 2-dimensional convolutional layers and fully connected linear layers in the end. The decoder part has a *mirrored* structure, starting from fully connected linear layers getting larger, followed by 2-dimensional transposed convolutional layers.
- Kernel size, stride, padding, the number of channels in convolutional layers of the encoder / decoder part are free parameters.
- The latent dimension is a free parameter.
- `ReLU` activation is used for convolutional / transposed convolutional layers of the encoder / decoder part.
- `Sigmoid` activation is used in the output layer of the decoder part.

Sub-problems:

Consider an autoencoder satisfying the specifications above and the following:

- The encoder has 4 convolutional layers and 1 linear layer. The number of output channels in the convolutional layers are 64, 32, 16, 2 (in that order). The linear layer's output size, the dimension of the latent space, is 3.
- The kernel size is all 3×3, and the padding is all 1. For the first two layers, the stride is set as 2, and for the rest two layers, the stride is 1.
- The decoder has 1 linear layer and 4 transposed convolutional layers. The linear layer mirrors the linear layer of the encoder. The number of output channels in the transposed convolutional layers are 16, 32, 64, 1 (in that order), respectively.
- The kernel size is all 3×3, and the padding is all 1. For the first two layers, the stride is set as 1, and for the rest two layers, the stride is 2.

a) Calculate the size of all layers' dimensions correctly. Calculate the total number of trainable parameters. (1 point)

b) Implement the autoencoder and the training algorithm written below by using PyTorch. Everything that is not specified is of your choice. (1.5 point)
   - Adam optimizer
   - training with mini-batches
   - default PyTorch initialization of network parameters
   - mean-squared error (MSE) loss

c) Train the autoencoder to learn a representation of the Fashion-MNIST dataset by using the *training* dataset until the validation loss becomes lower than $0.03 = 3e\text{-}2$. The validation loss here refers to the loss value *averaged over the whole training dataset for fixed model parameters*. Note that it is different from training loss, which is averaging the training loss of all batches, each of which is obtained with all different model parameters. Provide the model parameters after successful training. (2 point)

d) Illustrate the 3-dimensional latent representation of the *training* dataset learned in Task c) as a scatter plot using a different color for each of the 10 class labels. This is to visualize how distinguishable the samples of different classes in the latent space are. (1 point)

e) Make a 3-dimensional $12 \times 12 \times 12$ grid of linear-spaced values, which can cover the encoded images of the training dataset in the learned latent space. For instance, if all values in the plot of Task d) are roughly in $[-50, 5] \times [-10, 100] \times [10, 65]$, then make a grid of $\{-50, -45, -40, \ldots, 0, 5\} \times \{-10, 0, 10 \ldots, 90, 100\} \times \{10, 15, \ldots, 60, 65\}$. Then, pass the vectors of the grid through the decoder and illustrate the output images for those vectors. For example, it can be illustrated by 12 figures, each of which has 12×12 images. This is to observe how the vectors in the latent space are related to the reconstructed images. (1.5 point)

f) Calculate the test loss, the average MSE of the original images of the test dataset and the reconstructed images of them. Visualize the reconstructions of one test sample image of each class with comparison to the original image. (1.5 point)

Notes:

- What should be submitted is your calculations of Task a), the source code you implemented and used, the trained model of Task c) (use corresponding PyTorch functions for saving), the illustrated figures of Tasks d), e), f) you plotted, and the test MSE value you obtained.
- Be careful about tensors' size when you get the MSE loss. When the input two tensors' dimensions are different, the loss becomes wrong and the model does not converge.

Homework Problem 5: (DCGAN for Flowers102 dataset)

The goal is to implement a generative adversarial network (GAN) based on convolutional neural network (CNN) architectures using PyTorch to learn to generate Flowes102-dataset-like images.

Flowers102 dataset:
Flowers102 is a dataset of labeled RGB color images consisting of 102 flower categories. Each class consists of between 40 and 258 images. In total there are 8189 data samples split into training, validation and test samples. The Flowers102 dataset is generally challenging to learn as the images have large scale, pose and light variations. The figure below shows some samples.

The Flowers102 dataset can be downloaded and handled similar to the MNIST and Fashion-MNIST dataset using PyTorch (module `torchvision.datasets`). Use the code in the Jupyter notebook provided with this homework set to load the dataset in a form appropriate for this problem. If the dataset is not downloaded correctly, you can download it manually using the following link.

https://tud.link/dvyjrn

General architecture:
In this problem we aim to implement and train a deep convolutional GAN (DCGAN) as in Problem 22. It explicitly uses convolutional and convolutional-transpose layers in the discriminator and generator. The discriminator is made up of strided convolution layers, batch norm layers, and LeakyReLU activations. **Additionally, a so called spectral normalization is applied to normalize the weights of the convolutional layers to stabilize the training of the discriminator.** The generator is comprised of convolutional-transpose layers, batch norm layers, and ReLU activations.

Generator specifications:

- The input of the generator is a latent vector, that is randomly drawn from a standard normal distribution, where its dimension is a free parameter `LATENT_DIM`.
- The layers are composed of blocks of the form:
  convolutional-transpose layer $\longrightarrow$ batch norm layer $\longrightarrow$ ReLU activation.
- In the first convolutional-transpose layer the number of input channels is equal to the latent dimension and the number of output channels is a free parameter `CHANNEL_NUM`, where its value is chosen as a (large enough) power of 2.
- The number of channels in the convolutional-transpose layers is halved from layer to layer, a kernel size 4×4 is used, and the parameters for stride and padding are chosen such that the feature map size starts with 4×4 and doubles in each dimension from layer to layer.
- The output convolutional-transpose layer has 3 output channels (as we aim at generating color images), Tanh activation is used and no batch normalization.

Discriminator specifications:

- The discriminator input is the generators' output. The discriminator output is a scalar probability that the input is from the real dataset.
- The discriminator strided convolution layer architecture mirrors the generator architecture with the following modifications:
  - **The weights of the convolutional layers are normalized using spectral normalization.**
  - Leaky ReLU activation is used instead of ReLU activation.
  - At the input layer no batch normalization is used.
  - At the output layer sigmoid activation is used and no batch normalization.

Everything that is not explicitly specified is of your own choice.

Hint: The required spectral normalization of the discriminator weights can be achieved in PyTorch by wrapping `torch.nn.utils.spectral_norm()` around each convolutional layer. See the following link for more information.

https://pytorch.org/docs/stable/generated/torch.nn.utils.spectral_norm.html

Sub-problems: Provide the source code you implemented and used for all subsequent tasks.

a) (1 point) From the complete Flowers102 dataset select a subset containing all images of a predefined list of flower categories such as [`'passion flower'`, `'lotus'`, `'magnolia'`, `'water lily'`, ...]. The composed dataset is to be used subsequently to train the DCGAN and should contain roughly about 1600 images. Choose the flower categories to be selected at your own favor and provide the list. The following link gives an illustrated overview over all categories and category names.

   https://www.robots.ox.ac.uk/∼vgg/data/flowers/102/categories.html

   To relate category names and numerical class labels you can use the file `flower-categories.json` provided with the homework set (*Note:* Numerical image labels are {0,1,2,...,101} whereas the labels in `flower-categories.json` are {1,2,...,102}, i. e., they are shifted by 1.)
   Visualize sample images from the composed dataset in a grid layout containing one image of each of the selected categories. Provide the figure.

b) (1 point) Implement the generator network according to the given specifications such that the size of the generated images is 256x256 pixels. Implement the corresponding discriminator network.

c) (1.5 point) Implement a training algorithm for the specified DCGAN architecture. Within the training loop visualize the progression of the generator as follows. For a fixed batch of latent vectors generate after each training epoch a batch of images with the updated generator and visualize (a suitable number of) the generated images. Provide the figures. Furthermore, during the training track generator loss, discriminator loss, and discriminator output obtained for training images and generated images.

d) (1.5 point) Train the DCGAN architecture with the image dataset composed in Task a) until the generator produces output images looking sufficiently nice to your eyes. Provide the hyperparameters and the model parameters after successful training (use corresponding PyTorch functions for saving).
For comparison: After training the DCGAN with `CHANNEL_NUM` = 512 and `LATENT_DIM` = 128 using the *complete* dataset over 200 epochs the trained generator produces images as shown in the subsequent figure. The images can still be recognized as "fake", however, already show nice flowery structures. On the HPC JupyterHub the training took 1–2 minutes per epoch using GPU device.

e) (0.5 point) Illustrate the generator loss, the discriminator loss, and the discriminator output for the training images and generated images as recorded during the training. Based on the loss and output diagrams shortly explain, if the generator and discriminator behave as expected. Provide the diagrams and your explanations.

f) (1.5 point) Use the trained generator to generate images in two ways. Randomly sample a batch of latent vectors using the same batch size as used during the training. Then proceed as follows:
i) Feed the complete batch of latent vectors at once into the generator and illustrate the batch of generated images (or a subset of suitable size if the batch size is larger).
ii) Feed the latent vectors of the batch one-by-one into the generator and illustrate the generated images.
What do you observe? Is there a difference between i) and ii)? If yes, explain why and conclude how to use the generator correctly.
Provide the generated images together with the corresponding latent vectors and your explanations.

g) (1.5 point) Repeatedly use the trained generator and save/record your favorite generate images together with the latent vectors generating them.
Select your favorite two (sufficiently different) images, say I_1 and I_2, and the corresponding latent vectors, say z_1 and z_2. Then generate new images using the convex combination $z = \alpha z_1 + (1-\alpha) z_2$ as latent vector for $\alpha = 0.0, 0.1, 0.2, \ldots, 0.9, 1.0$. Illustrate the result and explain if the result is as expected.
Note: Be careful! For correct image generation take your observations of Task f) into account!
Provide the generated images together with the corresponding latent vectors and your explanations.