10907 Pattern Recognition

Lecturers
Prof. Dr. Ivan Dokmanić ⟨ivan.dokmanic@unibas.ch⟩

Tutors
Alexandra Flora Spitzer ⟨alexandra.spitzer@stud.unibas.ch⟩
Roman Fries ⟨r.fries@unibas.ch⟩
Cheng Shi ⟨cheng.shi@unibas.ch⟩
Vinith Kishore ⟨vinith.kishore@unibas.ch⟩
Valentin Debarnot ⟨valentin.debarnot@unibas.ch⟩

Assignment 5

Deadline: 07.12.2023
Total points: 6 (+1.5 bonus)

Math

Exercise 1 (Neural network and backpropagation, 1.5 point)**.**

Consider the one-hidden-layer neural network defined by the function:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}_2\sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2, \tag{1}$$

where $\boldsymbol{\theta} = \{\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{b}_1, \boldsymbol{b}_2\}$ represents the set of trainable parameters, and $\sigma(\cdot)$ is a element-wise non-linear activation function. These parameters are optimized using gradient descent update,

$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) - \gamma\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}(t)),$$

applied to a dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ with the loss function.

$$L(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N}(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i)^2.$$

- In a deep (more than one layer) neural network, we typically refer to $\boldsymbol{h}_1 = \sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$ as hidden (or middle) layer output. Given that input is $\boldsymbol{x} \in \mathbb{R}^K$ and the label is a scalar $y \in \mathbb{R}$, and assuming the hidden layer dimension is F, what are the dimensions of trainable parameters $\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{W}_1, \boldsymbol{W}_2$?
- For a single training datum $\boldsymbol{x}_1 = [1, 1]^\intercal$ and $y_1 = 1$, calculate the updated parameters $\boldsymbol{\theta}(2) = \{\boldsymbol{W}_1(2), \boldsymbol{W}_2(2), \boldsymbol{b}_1(2), \boldsymbol{b}_2(2)\}$ after the second step of gradient descent iteration. Use the initial values:

  $$\boldsymbol{W}_1(0) = [1, 1], \quad \boldsymbol{W}_2(0) = \boldsymbol{b}_1(0) = \boldsymbol{b}_2(0) = 0,$$

  with the activation function $\sigma(x) = \frac{1}{1+e^{-x}}$ and learning rate as $\gamma = 0.1$.

Code

In this numerical part, we will first warm up with a toy problem before addressing two standard problems of image processing, namely denoising and (blind) deblurring.

Exercise 2 (DL warm up, 1.5 point)**.**

Develop one-hidden-layer neural networks as described in **Exercise 1** with $\sigma(\cdot) = \mathrm{ReLU}(\cdot)$. Train this model using the following dataset, which consists of 9 data points sampled from the quadratic function $y = x^2$:

x	-1.2	-0.9	-0.6	-0.3	0	0.3	0.6	0.9	1.2
y	1.44	0.81	0.36	0.09	0.	0.09	0.36	0.81	1.44

Table 1: Training dataset sampled from $y = x^2$

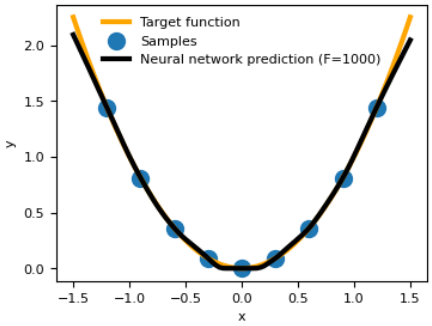

Figure 1: Sample plot showing the results of the one-hidden-layer neural network with $F = 1000$.

Illustrate the performance of your trained network by plotting the results for $x \in [-1.5, 1.5]$. Compare the results with different hidden layer dimensions: $F = 2, 5, 10, 50, 100$, and 500. For example, Figure 1 shows that the plot might look like for $F = 1000$. Be aware that the loss may not always diminish to zero. Evaluation will be based on these plots. Make sure your plots are clear, properly labeled, and include a legend if necessary.

Exercise 3 (Denoising 1 points)**.**

The three following exercises rely heavily on the material of the deep learning bootcamp. We strongly encourage the use of the Python functions defined during this bootcamp. Deep learning models can benefit significantly from GPU acceleration. You can check whether you are using a GPU by looking at the output of `torch.cuda.is_available()` (on your own machine or in Colab for instance) or you can use *sciCORE*.

We will use the MNIST dataset in all three exercises. You can use the function `data_generator` to generate and manipulate the dataset (the first call can be longer since it will download the images). Do not modify any function in the file `helper.py`. This will help you discover any potential problem with your code that could occur during grading (which would lead to a score of 0).

Given an image $u \in \mathbb{R}^{N \times N}$, denoising aims at recovering the image u from the observation of

$$y = u + \eta,$$

where η is additive white Gaussian noise. More precisely, each pixel (i, j) of the original image is degraded by independent random variable $\eta[i, j] \sim \mathcal{N}(0, \sigma^2)$. In this exercise, we assume that σ is fixed and equal to 0.5.

In the previous lectures and exercises we have seen how to use filtering for denoising. In this exercise, we are interested in denoising with a deep neural network. We will first focus on the very simple convolutional neural network (CNN) defined during the deep learning bootcamp. The neural network architecture can be found under the name `CNN_simple` in the file `helper.py`.

We also provide the function `add_noise.py` that adds Gaussian random noise to a batch of images. Details of the input dimension and arguments of the function can be found in the file `helper.py`. This function can be used to generate noisy images to train a neural network.

Your task: train the neural network implemented in the function `CNN_simple` to denoise images from the MNIST dataset.

Your trained network will be evaluated on another part of the MNIST dataset with additive Gaussian noise with $\sigma = 0.5$. We will quantify the quality of the estimation by using the Signal

to Noise Ratio (SNR), which is a common metric to asses the quality of reconstructed images compared to the true ones. You can check the SNR obtained with your trained network by running the file `denoising.py` on the MNIST test dataset. In this script, we compute the SNR between the denoised images and the original images. The higher the SNR, the closer the estimation is to the true image. We will grade this exercise according to the SNR obtained with your trained network.

Some rules to follow for successful grading:

- Assuming that your torch architecture is called `net` (obtained from `net=CNN_simple(image_size)`), you must save your model with the following command:

  `torch.save(net.state_dict(), "denoising.pt")`,

  and provide the file `denoising.pt` in your submission.
- Do not modify the neural network architecture in the file `helper.py`.

Once the model is saved, you can run the script `denoising.py` to check the results of your network. If the script runs (without modification) and you get a good SNR improvement, then you are guaranteed to get most of the points assigned to this exercise.

Hints:

- Try different losses: `https://pytorch.org/docs/stable/nn.html#loss-functions`. The most common losses for this application being `MSELoss` and `L1Loss`.
- Make sure to use the full training dataset and to train for enough epochs.
- Adjust the learning rate.

Exercise 4 (Deblurring 2 points)**.**

Given an image $u \in \mathbb{R}^{N\times N}$, the task of deblurring is to recover u from the observation of the convolution of u by a blur filter $h \in \mathbb{R}^{N\times N}$:

$$y = h \star u + \eta,$$

where η is an additive white Gaussian noise (as in the previous exercise). In this part, we consider smaller noise with $\sigma = 0.01$.

The blur kernel h is given by a Gaussian convolution filter with standard deviation $\sigma_{blur} = 1.5$ and normalize to sum to one. The blur kernel can be obtained using the function `Gaussian_blur` in the file `helper.py`. The convolution can be computed in Pytorch using the function `convolution_fft_torch` from `helper.py`.

Your task: train a neural network to perform deblurring. In this exercise, you can design any neural network architecture using Pytorch. As we have discussed during the deep learning bootcamp and the lectures, neural networks based on convolutions are usually preferred in image processing applications. In particular, autoencoder and Unet-like structures are particularly efficient.

We will evaluate your trained network on another part of the MNIST dataset while applying a Gaussian blur with $\sigma_{blur} = 1.5$ and additive Gaussian noise with $\sigma = 0.01$. We will quantify the quality of the estimation and grade this exercise with the SNR similarly to the previous exercise.

Some rules to follow for successful grading:

- Assuming that your torch architecture is called `net`, you must save your model with the following command:

```
model_scripted = torch.jit.script(net.cpu())
model_scripted.save("deblurring.pt")
```

and provide the file `deblurring.pt` in your submission.

- Once saved, the file `deblurring.pt` must **not exceed** 15mb. There could be an extra point for very light networks that performs well.

Once the model is saved, you can run the script `deblurring.py` to check the results of your network. If the script runs (without modification) and you get a good SNR improvement, then you are guarantee to get most of the points assigned to this exercise.

Hint:

- You can use the function `Upsample` from Pytorch to upsample an image (useful for autoencoder and Unet like architectures).
- Adding two upsampling layers at the end of the CNN defined in the previous exercise should already provide very accurate results.

Exercise 5 (Blind deblurring (bonus: 1.5 points))**.**

In this exercise, we observe a blurry-noisy image y such that:

$$y = h \star u + \eta,$$

where η is an additive white Gaussian noise. However, this time, we assume that the Gaussian filter h is not fixed but is defined with standard deviation uniformly chosen at random in the interval $[0.5, 2.5]$.

Your task: train a neural network (any architecture) that can deal with such incertitude about the blur operator.

Some rules to follow for successful grading:

- Assuming that your torch architecture is called `net`, you must save your model with the following command:

```
model_scripted = torch.jit.script(net.cpu())
model_scripted.save("deblurring_family.pt")
```

and provide the file `deblurring_family.pt` in your submission.

- Once saved, the file `deblurring_family.pt` must **not exceed** 15mb. There could be an extra point for very light networks that performs well.
- Name your submission with your real name.

Once the model is saved, you can run the script `deblurring_family.py` to check the results of your network.

For this exercise, we will run a leaderboard. The top 10 models in term of SNR (the higher the better) will get a 0.5 bonus point. The 10 fastest models that have a SNR improvement greater than 5 will get a +0.5 bonus point. To get the full bonus point, you should be in the top 10 submission for both the SNR and the running time with the **same** submission. A +0.5 extra bonus point will be awarded for submissions that perform best than the leader from last year in term of SNR.