As general comment, when doing practical sessions, please keep track of you work in a Word/Writer/GoogleDoc document. Trace also all codes you are creating. Large Language Models are not forbidden to get help as search tools, but do not use them to write the source code for exercises: do it by yourself!

A first network: MNIST

Exercice 1 Research, installation, first training

In this first practical session, we will employ the example code from PyTorch to handle the MNIST database.

? Conduct a thorough search about the MNIST (Modified National Institute of Standards and Technology) database. As seen in the lecture, this dataset has been widely used in machine learning.

? Get the « mnist.zip » file and unzip it in a suitable place. The dataset is small but need to be (automatically) downloaded.

? Install (Mini)conda if you do not have already it installed on your computer. Create a Conda environment specifically for executing the code contained within the "mnist.zip" archive. Got the pytoch website to install it regarding your computer hardware and software. Please also install *scikit-learn* and *matplotlib*.

? Execute the algorithm (found in "*main.py*") to ensure that everything is functional. Note that the code will initially download the MNIST corpus if it's not already present.

Exercice 2 Code analysis

We will now focus on examining the code itself and analyzing its contents in detail:

? Locate and distinguish between different sections of the code, including the neural network definition, the training procedure and the evaluation procedure. Analyze the structure of the neural network. What components does it contain? Examine how the training and evaluation procedures operate.

? Find the code related to the definition of the optimizer. Research information about the specific optimizer used in this implementation. Investigate how the optimizer is applied within the context of this code.

Exercice 3 Analysis of Loss Evolution During Training

We will now focus on the variations of the loss (actually, losses) that allow us to evaluate the evolution of a neural network's learning process. As a reminder, it is the loss that guides the network's learning in conjunction with backpropagation.

During training, the corpus is typically divided into 2 or 3 parts, named train, validation, and test respectively. The train part is the most important and largest, generally 80 to 90% of the corpus. This is the portion of data used to learn the model parameters. The validation part is smaller and contains data completely disjoint from the training part. For example, there should not be very similar images in the train and validation sets, or in the case of speech, the same person should not be present in both train and validation parts, otherwise the model would learn their voice. The purpose of the validation part is to show the model's performance on unseen data, and thus its generalization capacity. Finally, when there are 3 parts to the corpus, the test part is a portion for which we have the data for evaluation but not the labels.

This prevents cheating in scientific competitions where only the competition organizer knows these labels.

Thus, while training, one uses train and validation (the latter being called test in some corpora like MNIST where there are only 2 parts) to check the joint evolution of the loss on the training data on one side and on the validation data on the other. The evolution of losses should be as follows for a neural network during training:

- If the network is learning correctly, the training loss decreases and tends towards 0 (even if it oscillates). The lower the loss value, the more the network has learned from the training data.
- If the network generalizes correctly, the validation loss (or test loss in the case of MNIST) follows the same trend. If this loss starts to increase again, we say that the network is overfitting: it has "memorized" the training data and is not able to generalize properly.

In this exercice, we will focus on measuring the effects of modifications to MNIST training on the train and test losses.

? Take the original code found on Moodle (in the "*mnist.zip*" archive). Using the calculation done in the test function as inspiration, modify the train function to calculate the total loss on the training data. Also modify the test and train functions to be able to retrieve the train and test loss for each epoch. Run the training for all epochs and plot the 2 losses as curves, each on a separate graph. Comment on the obtained graphs.

? Reverse the train and test corpora of MNIST in the script to learn on the test corpus (smaller) and test on the train corpus (much larger). Plot the 2 loss curves as before and comment.

? Keep the corpus inversion but additionally remove the dropout functions in the Net class (the class containing the neural network definition). Start the training, research about dropout while the training is running, plot the losses and comment.

? Put the test and training corpora back to their original state. Leave the dropouts inactive. Run the training, plot the losses and comment.

? To synthesize the results of these 4 analyses, create 2 graphs. The first will contain the 4 training losses, the second the 4 test losses. Did you observe overfitting or not? Why? If yes, in which case(s)?

Exercice 4 Parameter Variations

? Identify the parameters that can be passed when calling the code. Consult the documentation to gain a comprehensive understanding of these parameters. Hypothesize about the purpose and effect of each parameter.

? Execute the code multiple times, varying the parameters systematically. Important: To isolate the effect of each parameter, only modify one parameter at a time while keeping others constant. Create a performance table (in spreadsheet like Excel, Calc or GoogleDoc). documenting the results obtained with different parameter values. Determine which combination of parameters yields the best solution. Try to analyze and explain why this particular combination produces optimal results. Is there any surprising results?

? Supplementary questions: is there any difference between CPU and GPU training? What are your global conclusion about these parameters?

Exercice 5 Performance comparison with ML alternatives

? Using the *time* module from python, computer the training time of the initial implementation. Compute also the mean inference time on all test images.

? Take a look at the "*main_resnet.py*" file. What has changed? Document yourself about this type of network. What does that mean for the network? Train this network to compare it with the initial version. What can you say/conclude looking at performance?

? What is the training time of this second network? What is the average inference time on one image? Compare with the initial network.

? After this first neural network architecture alternative, we will use an SVM as a classifier for MNIST data. Download the "*main_svm.py*" file, read it. How does it work?

? Compare performance and execution time in terms of training time and inference time. To do so, look at how report and confusion matrix are computer and drawn. Modify former code to do the same analysis. From the results, what can you say/conclude?

? We will now try to see the distribution of execution time for each of the called functions. To do this, when launching the program, instead of simply call "*python main.py...*", we will use "*python -m cProfile -s time main.py...*".

? Run the training for 1 epoch for initial program and look at the result obtained. What can you say about it?

? Make sure to save the loss values in a file when the program is launched. Vary the batch size and relaunch each time for 1 epoch. What do you observe for the loss? For the output of the code profiler?

? Do the same for the ResNet and the SVM. Compare results and comment.

Exercice 6 First step toward reproducible research

? Export your working environment using *conda* and *pip*, compare the 2 obtained files. Why do they differ?

? Create another environment using one of these files. Does it work?

? Share this environment with another student, preferably using a different operating system. Does it work?

? Open question: is there any way to be sure that it will work all times? What about Linux/Windows compatibility?

AudioMNIST: optimizing data loading

Exercice 7 Examining the AudioMNIST project

We will study a neural model for Audio MNIST, a database of people stating numbers from 0 to 9 in English (similar to MNIST but with audio files). this work is based on the source code proposed on the Kaggle platform: https://www.kaggle.com/code/jokekling/pytorch-study-audio. We will not train this model.

? Create a conda environment containing the following packages: *pip*, *pytorch*, *torchvision*, *torchaudio*, *pandas*, *numpy*, *matplotlib*. Activate this environment. If you are on Windows, run "*pip install pySoundFile*"; otherwise, run "*pip install sox*".

? Download "*AudioMNIST.zip*" from Moodle. Look at and listen to some examples from the "*recordings*" folder. You will find files named with the digit to be stated from 0 to 9, followed by the speaker's name, followed by the repetition number (from 0 to 50). We limit the Audio MNIST dataset to 3000 files for this practical work.

? Use the script "*ShowAudioFile.py*" to visualize some files (look at optional parameters for command line). The script uses (like the author on Kaggle) the MelSpectrogram. What does this correspond to? Which frequencies are represented in this graph?

? The scripts provided by the author automatically separate the corpus into train/test (block 1 of the online Jupyter notebook). This pratical section proposes a version of the script adapted to our data organization: "*SplitTrainTest_original.py*". Run this script and look at the content of the 2 generated CSV files. Compared to discussion in the lecture, what can you say about this train/test separation?

? We will study the blocks of the Jupyter Notebook. Could you describe what is done in the following blocks (from 2 to 15)?

? Block 16 presents the training and testing losses associated with the recognition score of digits (from 0 to 100%). Comment on this graph.

Exercice 8 Efficient load of audio files for training

? We will use the two Dataset loaders inspired byt the Kaggle project to load the data. Start with the provided code "*main_audioload.py*". Run this file in the AudioMNIST directory. What does it print? Check how long the main block takes to execute.

? Run the script a second time. How long did it take? Why?

? Comment out the line containing the "*print*" statement in the loop and check the execution time again. What do you observe? How can you resolve this?

? Profile the code with the tools seen previously. What can you say about the result with and without the line containing the "*print*" statement?

? Based on the Kaggle code, implement the same functionality with AudioDatasetSpectrogram. What do *ToSpec*, *ToSpectrogram*, and *ToDB* correspond to? Repeat the same performance tests as before.

? Change the *numworker* value. Try it. Check also the *multiprocessing_context* parameter and set it to *mp.get_context('spawn')*. How about performance loading result?

? We will cProfile tool, we will not be able to get information about data loading in multhreads or multiprocesses context. We can use the pytorch profiler. Download and execute the "*main_spectrogramload.py*". What information did you get? What can you conclude?

Exercice 9 Convert to pt and hdf5 files

? Look at how to convert the melspectrograms directly into ".pt" files. We want to regenerate directly the corpus in this format. What are the steps to do so? Do the conversion, and write a dataloader for this new version of the dataset.

? HDF5 files are files that can contain datasets and (numpy) arrays. This type of storage have some advantages. Operations on arrays like slicing can be done as its content. It is fast and can contains directly numpy arrays. Read the documentation about hdf5 https://docs.h5py.org/en/stable/quick.html#quick, convert you dataset in this format and create a dataloader for it.

? Evaluate performance of these dataloaders as you did for the former ones.