Computational Neuroscience

AY 2022-23

Laboratory Assignment 3 (LAB3)
TDNN and RNN networks

General Information

Solve the following assignment, whose completion is required to access the oral examination. Upload the assignments all-together in the Moodle platform of the course (once you have completed all the labs, not only this single one) as a compressed folder including one subfolder for each laboratory.

The subfolder for this lab should be called "LAB3_1" and should include the scripts and the other files as requested in the assignment below. You can organize the code as you wish, implementing all the helper functions that you need, provided that these are included in the subfolder and are appropriately called in the scripts.

Bonus track assignments are meant to be for those who finish early, but they are not formally required for completing the Lab Assignment.

Detailed instructions for submissions are given in the lecture note "Intro to the course".

The recommended programming language is MATLAB, but you are allowed to use Python (e.g., with NumPy, PyTorch, or TensorFlow).
If you use MATLAB, you will provide:

- the source files as a collection of the scripts (.m files) used to solve the assignment, or as a live script (.mlx file) with all the necessary code.
- the output data as a collection of binary .mat files, or in a single .mat file.

If you use Python, you will provide:

- the source files in one of these forms:
    - a collection of the scripts (.py files) used to solve the assignment, with a main script that calls all the necessary other scripts;
    - a Jupyter nootebook (.ipynb file) with all the necessary code.
- the output data as a collection of pickle or json files, or in a single pickle/json file
    - in this case it is required to include a script to load the saved data in the chosen format.

In both cases, the required figures can be provided in .pdf files or in a graphics format (e.g., png, jpg). When multiple images are required, you are allowed to generate a single file including multiple subplots (or pages, if appropriate).

Supporting Material

Matlab

Documentation on Time Series and Dynamic Systems: https://it.mathworks.com/help/deeplearning/time-series-and-dynamic-systems.html?s_tid=srchbrcm . See especially the API for functions timedelaynet (https://it.mathworks.com/help/deeplearning/ref/timedelaynet.html) and layrecnet (https://it.mathworks.com/help/deeplearning/ref/layrecnet.html)

A brief recap of the process for creating, customizing and training neural networks is in the file “LAB3_1-AdditionalMaterial_MATLAB.pdf” (available on the Moodle platform).

Matlab User’s Guide https://www.mathworks.com/help/index.html
Matlab documentation using the *help* command

Python (TF/Keras)

An online guide to create, customize and use RNN layers: https://keras.io/guides/working_with_rnns/
Basic API for recurrent models in Keras: https://keras.io/api/layers/recurrent_layers/

A brief recap of the process for creating, customizing and training recurrent neural networks with Keras is in the file “LAB3_1-AdditionalMaterial_Keras.pdf” (available on the Moodle platform).

Python (PyTorch)

Vanilla RNN module in PyTorch: https://pytorch.org/docs/stable/generated/torch.nn.RNN.html
Linear layer module in PyTorch: https://pytorch.org/docs/stable/generated/torch.nn.Linear.html

A brief recap of the process for creating, customizing and training a recurrent neural network with PyTorch is in the file “LAB3_1-AdditionalMaterial_PyTorch.pdf” (available on the Moodle platform).

Assignment 1: NARMA10 task

This task consists in predicting the output of a 10-th order non-linear autoregressive moving average (NARMA) system. Further information on this task can be found in the paper *Gallicchio, Claudio, and Alessio Micheli. "Architectural and markovian factors of echo state networks." Neural Networks 24.5 (2011): 440-456.*

The input of the system is a sequence of elements $x(t)$ randomly chosen according to a uniform distribution over [0, 0.5]. The output of the target system is computed as follows:

$$y(t) = 0.3\,y(t-1) + 0.05\,y(t-1)\sum_{i=1}^{10} y(t-i) + 1.5x(t-10)x(t-1) + 0.1$$

Given the input value $x(t)$, the task is to predict the corresponding value of $y(t)$.

- Import the dataset from the .csv file "NARMA10.csv" (available on the Moodle platform), where the first row represents the input and the second row represents the target output. Different columns represent different time-steps.
- Split the data into training (the first 5000 time steps), and test set (remaining time steps). Note that for model selection you will use the data in the training set, with a further split in training (first 4000 samples) and validation (last 1000 samples).
    - For the sake of problem understanding, you can try to first visualize the time-series data (using the matplotlib library in Python or the plot command in Matlab)

Remember: when training an RNN you want to make sure to keep the last hidden state of your RNN after the training session, and use it as initial hidden state of the validation session. Same applies when transitioning to the test session. In Keras you can set stateful=True, while in PyTorch you need to do it manually.

(1) Create and train a **Time Delay Neural Network (TDNN)** to solve the task, considering the fundamental hyper-parameters (e.g., length of the input delay, training function/optimizer, learning rate, number of training epochs, etc.).
(2) Perform a model selection (e.g., by grid search or random search) on the values of the hyper-parameters identified in the previous point. Select on the validation set the best hyper-parametrization, as the one with the smallest Mean Squared Error (MSE).
(3) Train the TDNN model with the selected hyper-parametrization on the whole training set, and evaluate its MSE on the training set and on the test set.
(4) Repeat steps (1)-(3) also for **Recurrent Neural Network (RNN)**, considering in this case the appropriate hyper-parameters (e.g., in this case you don't have an input delay line)

The output of the assignment should then consist in the following

- The source code.
    - include a specification of the set of hyper-parameters and the range of values considered for the model selection
- For both TDNN and RNN:
    - The model variable & the training information
        - e.g., in Matlab: the net structure and the training record structure

        - e.g., in Keras: the model and history variables (after training)
        - e.g., in PyTorch: the model, the optimiser, the training loss for each epoch, and final test loss
    - Training, validation and test Mean Squared Error
    - A plot with the target and output signals plotted together over time (i.e. on the X-axis there is time, on the Y-axis there is the signal value). <u>This type of plot is required for both training and test (2 plots for each type of NN).</u>
    - A plot of the learning curve. This consists in a plot containing – at least – the training error computed over the training epochs.
        - in Matlab, this information is available in the training record
        - in Keras, this information is available in the history variable
        - in PyTorch you have to store manually (e.g. in a python list) the computed loss at each epoch

Bonus-track Assignment 1: Mackey-Glass 17 task

Solve the same assignment as in the previous exercise, this time referring to the Mackey-Glass (MG) 17 task. The Mackey-Glass is a dynamical system defined by a time-delay differential equation that reads as follows

$$Q'(t) = \frac{\beta_0 Q(t-\tau)}{1+Q(t-\tau)^n} - \gamma Q(t).$$

The case of $\gamma = 0.1, \beta_0 = 0.2$ and $n = 10,$ with τ = 17, is often used in the literature as a benchmark for forecasting tasks, since it produces mildly chaotic dynamics. The task of interest for this assignment is a next-step prediction task (autoregressive, a particular case of transduction) on the MG time series.

The data for this task is available in the file “MG17.csv” (available on the Moodle platform), and it consists of one single row containing the values of the Q(t) time-series for all the time-steps in the different columns. The task consists in predicting 1 step in the future the Mackey-Glass dynamical system, i.e. predicting the value Q(t+1), just looking at Q(t) (or at a tapped delay line of order s, i.e. Q(t),...,Q(t-s), for a TDNN).
Before splitting the data into training, validation and test splits, you need to organize it into input and target information. Specifically, the input should consist of the time-series values from the first time-step to the second-to-last, and the target in the time-series value from the second time-step to the last.

This assignment requires the same output items as in the previous one.

Bonus-Track Assignment 2: Sequential MNIST classification task

The MNIST dataset is a common dataset used in computer vision. It consists of 60000 images of 28x28 pixels of handwritten digits from 0 to 9. A strategy is to flatten the 28x28 pixels into 784 dimensional vectors and feed them as input to a multilayer perceptron. The Sequential MNIST task (sMNIST) is a challenging task devised to test the ability of an RNN model to learn long-term dependencies.
The sMNIST consists of considering the flattened MNIST vectors as a time series of 784 time steps, i.e. we feed an RNN with one pixel per time step starting from the top left of the image ending to the bottom right. **After 784 time steps** the RNN provides a 10-dimensional output containing the softmax probabilities of the input being one of the 10 digits.
NB: you don't have to produce an output for each time step, but rather you just need to use the last hidden state to perform the classification. The rationale is that the RNN, after "listening" to the whole time series, has encoded in its hidden state the information necessary to classify the time series.
If you're having fun, you can permute the flattened vectors (use the same permutation for all the instances of the dataset) before sending them to the RNN. This is called the Permuted Sequential MNIST task. Further details can be found in the ArXiv paper "Ceni, Andrea. "Random orthogonal additive filters: a solution to the vanishing/exploding gradient of deep neural networks." *arXiv preprint arXiv:2210.01245* (2022)"

Fig. 4. **Left.** The first training sample of the MNIST dataset is a 28×28 pixel image representing an handwritten digit 5. **Top right.** The 28×28 pixel image representing the digit 5 is flattened in a $784 = 28^2$ sequence starting from the top left corner, proceeding from left to right and from top to bottom, until the bottom right corner. The sequence is plotted in 8 rows for a better visualisation. **Bottom right.** The resulting sequence is then permuted according to a predetermined permutation.

How to get the MNIST dataset:

MNIST data setup

We will use the classic MNIST dataset, which consists of black-and-white images of hand-drawn digits (between 0 and 9).

We will use pathlib for dealing with paths (part of the Python 3 standard library), and will download the dataset using requests. We will only import modules when we use them, so you can see exactly what's being used at each point.

```
from pathlib import Path
import requests

DATA_PATH = Path("data")
PATH = DATA_PATH / "mnist"

PATH.mkdir(parents=True, exist_ok=True)

URL = "https://github.com/pytorch/tutorials/raw/main/_static/"
FILENAME = "mnist.pkl.gz"

if not (PATH / FILENAME).exists():
        content = requests.get(URL + FILENAME).content
        (PATH / FILENAME).open("wb").write(content)
```

This dataset is in numpy array format, and has been stored using pickle, a python-specific format for serializing data.

```
import pickle
import gzip

with gzip.open((PATH / FILENAME).as_posix(), "rb") as f:
        ((x_train, y_train), (x_valid, y_valid), _) = pickle.load(f, encoding="latin-1")
```

The dataset can also be downloaded in pickle format from the following link:
https://www.dropbox.com/s/wbbjytv2dbi7oqj/MNIST_dataset.p?dl=0

Bonus-Track Assignment 3: BackPropagation Through-Time algorithm from scratch

Implement from scratch, e.g. with MATLAB or Numpy, the BackPropagation Through-Time algorithm for training an RNN to solve the above tasks on time-series.

Notice that the BPTT derivation for the sequence transduction case (i.e., when you have an output at each input time-step, rather than an output at the end of the time-series) was left as an exercise.