EE559 – Practical Session 2

François Fleuret

https://fleuret.org/ee559/

[version of: October 4, 2018]

Introduction

The objective of this session is to continue practicing with tensors, deal with a real data-set, and get a feeling of how good/bad are the k-nearest neighbor rule and the PCA dimension reduction on MNIST and CIFAR10.

The questions should be answered by writing a source file and executing it by running the `python` command in a terminal, with the source file name as argument.

Both can be done from the main Jupyter window with.

- "New" → "Text file" to create the source code, or selecting the file and clicking "Edit" to edit an existing one.
- "New" → "Terminal" to start a shell from which you can run python.

Another option is to connect to the VM on port 2022 on the host with a SSH client such as PuTTY[1].

The source should start with

```
import torch
from torch import Tensor
import dlc_practical_prologue as prologue
```

to use the functions provided in the prologue. You are of course free to do without it.

You can get information about the practical sessions and the provided helper functions on the course's website.

https://fleuret.org/ee559/

1 Nearest neighbor

Write a function that gets a training set and a test sample and returns the label of the training point the closest to the latter.

More precisely, write:

```
def nearest_classification(train_input, train_target, x):
```

[1] https://www.putty.org/

where

- `train_input` is a 2d float tensor of dimension $n \times d$ containing the training vectors,
- `train_target` is a 1d long tensor of dimension n containing the training labels,
- `x` is 1d float tensor of dimension d containing the test vector,

and the returned value is the class of the train sample closest to x for the L^2 norm.

Hint: The function should have no python loop, and may use in particular `torch.mean`, `torch.view`, `torch.pow`, `torch.sum`, and `torch.sort` or `torch.min`. My version is 164 characters long.

2 Error estimation

Write a function

```
def compute_nb_errors(train_input, train_target, test_input, test_target,
                      mean = None, proj = None):
```

where

- `train_input` is a 2d float tensor of dimension $n \times d$ containing the train vectors,
- `train_target` is a 1d long tensor of dimension n containing the train labels,
- `test_input` is a 2d float tensor of dimension $m \times d$ containing the test vectors,
- `test_target` is a 1d long tensor of dimension m containing the test labels,
- `mean` is either `None` or a 1d float tensor of dimension d,
- `proj` is either `None` or a 2d float tensor of dimension $c \times d$,

that subtracts `mean` (if it is not `None`) from the vectors of both `train_input` and `test_input`, apply the operator `proj` (if it is not `None`) to both, and returns the number of classification errors using the 1-nearest-neighbor rule on the resulting data.

Hint: Use in particular `torch.mm`. My version is 487 characters long, and it has a loop (the horror!)

3 PCA

Write a function

```
def PCA(x):
```

where `x` is a 2d float tensor of dimension $n \times d$, which returns a pair composed of the 1d mean vector of dimension d and the PCA basis, ranked in decreasing order of the eigen-values, as a 2d tensor of dimension $d \times d$.

Hint: The function should have no python loop, and use in particular `torch.eig`, and `torch.sort`. My version is 275 characters long.

4 Check that all this makes sense

Compare the performance of the 1-nearest neighbor rule on data projected either a 100d random subspace (*i.e.* using a basis generated with a normal) and using the PCA basis for different dimensions (e.g. 3, 10, 50, 100).

Compare also the performance between MNIST and CIFAR. Does all this make sense?