EE–559: Practical Session 1

François Fleuret

https://fleuret.org/dlc/

February 17, 2018

Draft, do not distribute

Introduction

The objective of this session is to practice with basic tensor manipulations in pytorch, to understand the relation between a tensor and its underlying storage, and get a sense of the efficiency of tensor-based computation compared to their equivalent python iterative implementations.

You can get an information sheet about the practical sessions and provided helper functions at

https://fleuret.org/dlc/dlc-info-sheet.pdf

1 Multiple views of a storage

Generate the matrix

```
1 2 1 1 1 1 2 1 1 1 1 2 1
2 2 2 2 2 2 2 2 2 2 2 2 2
1 2 1 1 1 1 2 1 1 1 1 2 1
1 2 1 3 3 1 2 1 3 3 1 2 1
1 2 1 3 3 1 2 1 3 3 1 2 1
1 2 1 1 1 1 2 1 1 1 1 2 1
2 2 2 2 2 2 2 2 2 2 2 2 2
1 2 1 1 1 1 2 1 1 1 1 2 1
1 2 1 3 3 1 2 1 3 3 1 2 1
1 2 1 3 3 1 2 1 3 3 1 2 1
1 2 1 1 1 1 2 1 1 1 1 2 1
2 2 2 2 2 2 2 2 2 2 2 2 2
1 2 1 1 1 1 2 1 1 1 1 2 1
```

with no python loop.

Hint: Use several `torch.narrow`, and `torch.fill_`.

2 Eigendecomposition

Without using python loops, create a square matrix M (a 2d tensor) of dimension 20×20, filled with random Gaussian coefficients, and compute the eigenvalues of $M^{-1}\,\mathrm{diag}(1,\dots,20)\,M$.

Hint: Use `torch.arange`, `torch.diag`, `torch.mm`, `torch.inverse`, `torch.normal_` and `torch.eig`.

3 Flops per second

Generate two square matrices of dimension 5000×5000 filled with random Gaussian coefficients, compute their product, measure the time it takes, and estimate how many floating point products have been executed per second (should be in the billions or tens of billions).

Hint: Use `torch.normal_`, `torch.mm`, and `time.perf_counter`.

4 Playing with strides

Write a function `mul_row`, using python loops, that gets a 2d tensor as argument, and returns a tensor of same size, equal to the one given as argument, with the first row kept unchanged, the second multiplied by two, the third by three, etc. For instance:

```
>>> m = Tensor(4, 8).fill_(2.0)
>>> m

    2     2     2     2     2     2     2     2
    2     2     2     2     2     2     2     2
    2     2     2     2     2     2     2     2
    2     2     2     2     2     2     2     2
[torch.FloatTensor of size 4x8]

>>> mul_row(m)

    2     2     2     2     2     2     2     2
    4     4     4     4     4     4     4     4
    6     6     6     6     6     6     6     6
    8     8     8     8     8     8     8     8
[torch.FloatTensor of size 4x8]
```

Then, write a second version named `mul_row_fast`, using tensor operations.

Apply both versions to a matrix of size $10,000 \times 400$ and measure the time each takes (there should be more than two orders of magnitude difference).

Hint: Use `torch.arange`, `torch.view`, `torch.expand_as`, `torch.mul`, and `time.perf_counter`.