Deep Learning
Practical Session 1

François Fleuret

https://fleuret.org/dlc/

December 20, 2020

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

https://fleuret.org/dlc/

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 `torch.full`, and the slicing operator.

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:

$$M^{-1} \underbrace{\begin{bmatrix} 1 & 0 & \dots & \dots & 0 \\ 0 & 2 & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & 19 & 0 \\ 0 & \dots & \dots & 0 & 20 \end{bmatrix}}_{\text{diag}(1,\dots,20)} M$$

Hint: Use `torch.empty`, `torch.normal_`, `torch.arange`, `torch.diag`, `torch.mm`, `torch.inverse`, 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.empty`, `torch.normal_`, `torch.mm`, and `time.perf_counter`.

4 Playing with strides

Write a function `mul_row`, using python loops (and not even slicing operators), that gets a 2d tensor as argument, and returns a tensor of same size, whose first row is identical to the first row of the argument tensor, the second row is multiplied by two, the third by three, etc.

For instance:

```
>>> m = torch.full((4, 8), 2.0)
>>> m
tensor([[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.]])
>>> mul_row(m)
tensor([[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.]])
```

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

Apply both versions to a matrix of size $1,000 \times 400$ and measure the time each takes (there should be more than two orders of magnitude difference).

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