EE559 - Practical session

Francisco Massa
Facebook AI

May 15th, 2019

1 - Higher order interactions

Given `x` a tensor of shape `N`, and `J` a tensor of shape `[N, N, N]`, implement the following equation.

$$f(x_1, \dots, x_n) = \sum_{i=1,j=1,k=1}^{n} J_{i,j,k} x_i x_j x_k$$

The implementation should have no for loops, and return a single scalar..
Compare the your implementation runtime against a naive (for-loop) one.

2 - Intersection over union

The Intersection over Union (IoU) is a metric that measures the overlap between two binary shapes, and it is generally used for object detection systems.
The IoU is calculated as a ratio of area of the intersection and area of the union between the two shapes.

2.1 IOU OF TWO MASKS

Given two binary masks `mask_a` and `mask_b`, both of shape `[H, W]` , compute the IoU between `mask_a` and `mask_b`.
Hint: you should use the bitwise AND `&` and bitwise OR `|` operators, as well as `torch.sum`.

2.2 IOU FOR TWO SETS OF MASKS

Given `masks_a` a tensor of shape `[N, H, W]` and `masks_b` a tensor of shape `[K, H, W]` , compute the `[N, K]` matrix of pairwise IoU between each set of masks in `masks_a` and `masks_b`.
Hint: you should use the implementation of **2.1** and using for loops is the recommended approach for this exercise.

2.3 OPTIMIZED IOU FOR TWO SETS OF MASKS

The implementation in **2.2** has two nested for loops. Re-implement it so that it doesn't use loops.
Now compare the runtime between both implementations, for `masks_a` of size `[10, 100, 150]` and `masks_b` of size `[20, 100, 150]`. How much speedup can you obtain?
Hint: use broadcasting to batch the operations together

3 - Anti-diagonal matrix

An **anti-diagonal matrix** is a square matrix where all the entries are zero except those on the diagonal going from the lower left corner to the upper right corner. Here is an example of an anti-diagonal matrix:

```
tensor([[0, 0, 0, 6],
        [0, 0, 3, 0],
        [0, 1, 0, 0],
        [5, 0, 0, 0]])
```

Given a tensor x of N elements, return the matrix containing x in its anti-diagonal.
Hint: you should use advanced indexing on the left-hand side of the operation.

4 - 1d nearest neighbor upsampling

Given a tensor `x` of size `N` and an upsampling factor `k`, the nearest neighbor upsampling returns a tensor of size `floor(N * k)`, where each element of `x` is repeated `k` times (for integer `k`).
As an example

```
x = tensor([0.7330, 0.5974, 0.5389])
y = nearest_upsample(x, k=3)
# y is
tensor([0.7330, 0.7330, 0.7330, 0.5974, 0.5974, 0.5974, 0.5389, 0.5389, 0.5389])
```

For non-integer upsampling factor, some of the elements will be repeated more often than others.

```
x = tensor([0.6621, 0.3664, 0.2026])
y = upsample_nearest(x, 2.9)
# y is
tensor([0.6621, 0.6621, 0.6621, 0.3664, 0.3664, 0.3664, 0.2026, 0.2026])
```

4.1 INTEGER UPSAMPLING FACTOR

Write a function `nearest_upsample` that supports integer k.
Hint: you can use `reshape`, `expand`, or advanced indexing

4.2 ARBITRARY UPSAMPLING FACTOR

Generalize `nearest_upsample` to work with non-integer `k` . The output shape should be `floor(N * k)`.
Hint: use advanced indexing and `torch.arange` to obtain indices