NumPy Array vs PyTorch Tensor

NumPy is the most widely used library for scientific and numeric programming in python. It provides similar functionality and a similar API to that provided by PyTorch; however, it does not support using GPU or calculating gradients, which are both critical for deep learning.

Python is slow compared to many languages. Anything fast in Python, NumPy, or PyTorch is likely to be a wrapper for a compiled object written (and optimized) in another language-specially, C. In fact, NumPy arrays and PyTorch tensors can finish computations many thousands of times faster than using pure Python.

NumPy arrays:

A NumPy array is a multidimensional table of data, with all the items of same type. Since that can be any type at all, they can even be arrays of arrays, with the innermost arrays potentially being different sizes-this is called a jagged array. If the items are all of simple type such as integer or float, NumPy will store them as a compact C data structure in memory. This is where NumPy shines. NumPy has a wide variety of operators and methods that can run computations on these compact structures at the same speed as optimized C, because they are written in optimized C.

PyTorch tensors:

A PyTorch tensor is nearly the same thing as a NumPy array, but with an additional restriction that unlocks additional capabilities. It's the same in that it, too, is a multidimensional table of data, with all items of the same type. However, the restriction is that a tensor cannot use just any old type—it has to use a single basic numeric type for all components. As a result, a tensor is not as flexible as a genuine array of arrays. A PyTorch tensor cannot be jagged. It is always a regularly shaped multidimensional rectangular structure.

Major difference:

The vast majority of methods and operators supported by NumPy on these structures are also supported by PyTorch, but PyTorch tensors have additional capabilities. One major capability is that these structures can live on the GPU, in which case their computation will be optimized for the GPU and can run much faster (given lots of values to work on). PyTorch can automatically calculate derivatives of these operations, including combination of operations.

```
data = [[1,2,3],[4,5,6]]

arr = array(data)

tns= tensor(data)

arr  # numpy
```

```
array([[1,2,3],
       [4,5,6]])
tns    #tensor
tensor([[1,2,3],
        [4,5,6])
```