Introduction

In this lesson, we'll learn how to maximize the speed using NumPy!

This chapter explains the basic anatomy of NumPy arrays, especially regarding the memory layout, view, copy and the data type. They are critical notions to understand if you want your computation to benefit from NumPy philosophy.

Let's consider a simple example where we want to clear all the values from an array which has the data type `np.float32`. How does one write it to **maximize speed**? The below syntax is rather obvious (at least for those familiar with NumPy) but the above question asks to find the fastest operation.

```
import numpy as np
Z = np.ones(4*1000000, np.float32) #create an array of ones of size 4 *1000000
print(Z)
Z[...] = 0 #clear the array,sets every value to 0
print(Z)
print(Z.dtype)#prints the datatype of Z
```

RUN SAVE RESET

Output 1.076s

```
[1. 1. 1. ... 1. 1. 1.]
[0. 0. 0. ... 0. 0. 0.]
float32
```

If you look more closely at both the `dtype` and the size of the array, you can observe that this array can be casted (i.e. viewed) into many other "compatible" data types. By compatible, I mean that `Z.size * Z.itemsize` can be divided by the new `dtype itemsize`.

main.py
tools.py

```
import numpy as np
from tools import timeit #get timeit from tools.py(custom module)
Z = np.ones(4*1000000, np.float32) #create an array of size 4*10000000 np.float

print("np.float16:")
#time required to view array as np.float16
timeit("Z.view(np.float16)[...] = 0", globals())

print("np.int16:")
#time required to view array as np.int16
timeit("Z.view(np.int16)[...] = 0", globals())

print("np.int32:")
#time required to view array as np.int32
timeit("Z.view(np.int32)[...] = 0", globals())

print("np.float32:")
#time required to view array as np.float32
timeit("Z.view(np.float32)[...] = 0", globals())

print("np.int64:")
#time required to view array as np.int64
timeit("Z.view(np.int64)[...] = 0", globals())

print("np.float64:")
#time required to view array as np.float64
timeit("Z.view(np.float64)[...] = 0", globals())
```

RUN SAVE RESET

Output 6.296s

```
np.float16:
100 loops, best of 3: 982 usec per loop
np.int16:
100 loops, best of 3: 994 usec per loop
np.int32:
100 loops, best of 3: 969 usec per loop
np.float32:
100 loops, best of 3: 960 usec per loop
np.int64:
```

Here timeit is a custom function used. Interestingly enough, the obvious way of clearing all the values is not the fastest. The total number of CPU cycle to execute each above instruction are 100 but the two instruction take less time per loop. By casting the array into a larger data type such as `np.float64`, we gained a **25% speed factor**. But, by viewing the array as a byte array (`np.int8`), we gained a **50% factor**. The reason for such speedup is to be found in the internal NumPy machinery and the compiler optimization.

Solve this Quiz !

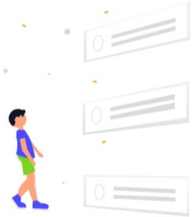
How can you increase the speed factor for clearing data from an array(setting all values in an array to 0)?

```
Z = np.ones(4*1000000, np.float32)
```

Correct Answer

A) `timeit("Z.view(np.float64)[...] = 0", globals())`

B) `timeit("Z.view(np.float16)[...] = 0", globals())`



You skipped the question. Would you like to try again?

RETAKE QUIZ

This simple example illustrates the philosophy of NumPy. Let's move on to the next lesson to learn memory layouts.