Midterm_W220205

February 8, 2022

1 Academic Integrity Statement

As a matter of Departmental policy, **we are required to give you a 0** unless you **type your name** after the following statement:

> *I certify on my honor that I have neither given nor received any help, or used any non-permitted resources, while completing this evaluation.*

[Jiaxin Luo]

1.0.1 Partial Credit

Let us give you partial credit! If you're stuck on a problem and just can't get your code to run:

First, **breathe**. Then, do any or all of the following:

1. Write down everything relevant that you know about the problem, as comments where your code would go.
2. If you have non-functioning code that demonstrates some correct ideas, indicate that and keep it (commented out).
3. Write down pseudocode (written instructions) outlining your solution approach.

In brief, even if you can't quite get your code to work, you can still **show us what you know.**

1.0.2 Rules

1. All type checking should use `isinstance` rather code of the form `type(x)==list`. If you don't remember how this works, refer to class notes or give it a quick google.

2. Whenever reasonably possible, for full credit you should use `list comprehensions` rather than modifying / creating items of a list/array one item at a time.

3. Operations like `x + y` or `2*x` should not modify the original `TwoDArray` instance `x`. This should not be a problem if you're using list comprehensions in those methods. (This rule will make sense after you read the rest of the problem.)

4. You can significantly shorten your code by having your methods call other methods. Therefore, in order to receive full credit, you should do this when reasonably possible. In particular, in my solution, the add magic method calls itself in several places. (This is allowable because there are a bunch of if statements.) If you are confused by this last idea, it might help to look at the example of a function calling itself at the bottom of the exam.

5. All lines of code should be 80 or fewer characters. You should also check that your code is visible when you print your pdf.
6. Solutions which do not, for example, use list comprehensions or have methods call other methods when appropriate, will receive partial credit. If you are struggling, a perfectly valid strategy is to first write a down a "less good" solution that works, do the rest of the exam, and then try to make your "less good" solution into a better solution.

2 Problem 1

In this problem, we will implement a `TwoDArray` class that is meant to mimic (some of) the functionality of a two-dimensional numpy array. You should not use `numpy` at any point. If you solutions uses `numpy`, you receive no credit for the part of the problem where `numpy` is used.

Before we proceed: Let's shortly recap two-dimensional arrays and the numpy implementation of them. We can think of two-dimensional arrays as tables of numbers: we refer to them as being two-dimensional because each number in the table has both a row and column index. They are very useful as it is often natural to store data in tabular form. Furthermore matrices, which play a very important role in many areas of Math, Computer Science and Engineering, are also represented using them! Here is an example in numpy.

[1]:
```
#Run this cell
import numpy as np

my_2d_array = np.array([[1,2],[3,4],[5,6]])
print("This is my_2d_array:")
print(my_2d_array)
print("This array has shape " + str(my_2d_array.shape) + " as there are " +␣
 ↪str(my_2d_array.shape[0]) +  " rows and " + str(my_2d_array.shape[1]) + "␣
 ↪columns.")
print("The entry of the array with row index " + str(1) + " and column index "␣
 ↪+ str(1) + " has value " + str(my_2d_array[1,1]))
```

```
This is my_2d_array:
[[1 2]
 [3 4]
 [5 6]]
This array has shape (3, 2) as there are 3 rows and 2 columns.
The entry of the array with row index 1 and column index 1 has value 4
```

Two-dimensional arrays as implemented in numpy support a number of operations including scalar multiplication and array addition.

[2]:
```
#Run this cell
print("If you multiply a two-dimensional array by a number (or scalar) then␣
 ↪each entry is multiplied by that number:")
print(2*my_2d_array)

print("If you add two-dimensional arrays their entries are summed:")
```

```python
print(my_2d_array + np.ones((3,2)))
```

```
If you multiply a two-dimensional array by a number (or scalar) then each entry
is multiplied by that number:
[[ 2  4]
 [ 6  8]
 [10 12]]
If you add two-dimensional arrays their entries are summed:
[[2. 3.]
 [4. 5.]
 [6. 7.]]
```

Your task in this problem will be to implement a class, using the template provided, which supports the creation of two-dimensional arrays, scalar multiplication, array addition and iteration. (In the above example, the first array is an array of ints and the second is an array of floats, which is why there is a decimal point there. You don't need to worry about this for this problem.)

2.1 Part A: Give your class an `__init__` method. - 10 points

I have already written part of this method for you. What you should do is make sure that the variable `array` is a valid input. This means you should a) Make sure that `array` is a list of lists. b) Make sure that each of the inner lists has the same length. You do not need to check for anything else.

If these conditions are not met you should raise the appropriate error(s). By this I mean it is up for you to decide if you should raise a `TypeError`, `KeyError`, `IndexError`, `ValueError`, `ZeroDivisionError`, etc. When raising the error, you should also give an informative error message.

With actual numpy arrays, all numbers are required to be the same datatype. You don't have to worry about that here. You can also assume that all numbers are `int`s or `float`s so you only have to worry about checking that the variable `array` is a list of lists.

3 Part B: Enable Scalar Multiplication - 10 points

Make scalar multiplication work the same way as with numpy arrays. For example, if `x=TwoDArray([[1,2][3,4]])`, then `2*x` should be `TwoDArray([[2,4][6,8]])`. For full credit, you solution should be four lines or fewer (not including `def`). You do not need to have comment or docstrings for this part. You also do not have to do any type checking for this part.

4 Part C: Enable Addition - 30 points

You should write a method so that if `x` and `y` are two `TwoDArray`s, the code `x+y` works correctly.

a) Your code should check that `x` and `y` are `TwoDArray`s and if not, raise an appropriate error with a short error message.

b) If `x` and `y` have the same shape, your method should add them element-wise and return a new `TwoDArray`, just like numpy operations.

c) If `x` and `y` have different shapes, your method should try to apply the **some** of the same rules of broadcasting as with numpy arrays. See https://numpy.org/doc/stable/user/basics.broadcasting.html or https://jakevdp.github.io/PythonDataScienceHandbook/02.05-computation-on-arrays-broadcasting.html#Rules-of-Broadcasting for review. In particular, implement broadcasting such that for example, `TwoDArrays` with shape (1, 3) or (4, 1) can be added to a `TwoDArray` with shape (4, 3).

Your code does **not** need to be able to handle the following broadcasting cases such as: - shape (3,) to (4, 3). We are only using two-dimensional arrays in `TwoDArray`. - shape (1, 1) to (4, 3). This works in numpy, but you do not need to consider it for this exam. - adding arrays with shapes (1, 3) and (4, 1). This works in numpy, but you do not need to consider it for this exam.

In order to apply the rules of broadcasting, my solution uses a lot of `if`/`elif`/`else` statements. My function also calls itself where appropriate. This allows me to avoid copying/pasting code. For full credit, your solution should also do this since copy/pasting code is considered bad style. However, if you do copy/paste code and make small edits, you will receive partial credit.

Add a short docstring and comment as appropriate.

Note: This will likely be the most time consuming portion of the exam. If your solution works when the arrays are the same size, but doesn't implement broadcasting you will receive partial credit. If you are struggling on this part, a good strategey is a) first do this problem without broadcasting b) then do the rest of the exam c) then come back to this part and implement broadcasting.

5 Part D: Enable Subtraction - 10 points

Write a method so that subtraction works in a way analogous to addition. You do not have to write comments for this line. However, for full credit you solution should only be one line (not including `def`). Hint: Use your other functions.

6 Part E: Enable Iteration - 20 points

Make your `TwoDArrays` iterable. Your iterator should go through the array one number at a time. It should first go across the top row, then across the next row, then across the next row, etc. For example if `x=TwoDArray([[1,2],[3,4]])`, then

```
for t in x:
    print(t)
```

should print

```
1
2
3
4
```

You can not use generator for this problem.

[7]:
```
class TwoDArray(list):
    #part A
    def __init__(self, array):
        """
        Initialize an instance variable array with a list of lists that have␣
↪the same length
        Args:
            array: a list variable
        Returns:
            None
        """

        ##########################
        if not isinstance(array, list) & all(isinstance(i, list) for i in␣
↪array):
            raise TypeError("This function is designed to work only with list␣
↪of lists.")
        elif len(array)>1 and all(len(array[0]) != len(i) for i in array[1:]):
            raise ValueError("All lists need to have the same length.")
        ############################

        #this line is written for you
        self.array=array

    #I am giving you these for free to save you time
    def __str__(self):
        return '[' + "\n ".join([str(arr) for arr in self.array]) + ']'

    def shape(self):
        return (len(self.array), len(self.array[0]))


    #part B
    def __rmul__(self, c):
        m, n = self.shape()

        #################################
        return [[c*self.array[row][col] for col in range(n)] for row in␣
↪range(m)]
        #################################

    #Part C
    def __add__(self, other):
        """
        Add two TwoDArrays as two numpys followed by the rules of broadcasting.
        When the inputs have the same shape, add them directly.
        When the first input has the lists with length of 1, broadcast it to␣
↪the same size as the other input, vice versa.
```

```
        When the first input has only one input, broadcast it to the same size␣
↪as the other input, vice versa.

        Args:
            other: an instance of TwoDArray
        Returns:
            the addition of two TwoDArrays with or without broadcasting
        """

        ##########################
        result = TwoDArray([])      # creat a return result of TwoDArray type
        #check that self and other are TwoDArrays
        if not (isinstance(self, TwoDArray) & isinstance(other, TwoDArray)):
            # if not, raise an appropriate error with a short error message.
            raise TypeError ("Two instances should be TwoDArrays!")
        ############################

        ############ DO NOT CHANGE ############
        #dummy variables for shorter lines
        sa, oa = self.array, other.array
        #array dimesnions
        m, n = self.shape()
        m1, n1 = other.shape()
        ########################################



        if m==m1 and n==n1:       # same shape, add items at the same position
            result.array = [[sa[r][c] + oa[r][c] for c in range(n)] for r in␣
↪range(m)]
        elif m==m1 and n==1:      # sa is m-by-1 matrix, boardcast it to m-by-n1
            sa = TwoDArray([(sa[r]*n1) for r in range(m)])
            result = sa.__add__(TwoDArray(oa))
        elif m==1 and n==n1:      # sa is 1-by-n matrix, boardcast it to␣
↪m1-by-n
            result = TwoDArray(oa).__add__(TwoDArray(sa*m1))
        elif (m==m1 and n1==1) or (m1==1 and n==n1): #oa is 1-by-n or m-by-1
            result = TwoDArray(oa).__add__(TwoDArray(sa)) # reverse the order␣
↪when calling itself
        else:
            result = None
            print("This type of addition can work in numpy by broadcasting.")

        return result
    #part D
    def __sub__(self, other):
        return self.__add__(TwoDArray([[-num for num in lis] for lis in other.
↪array]))
```

```
    #Part E
    #I have written this method for you to save time
    def __iter__(self):
        return TwoDArrayIterator(self)

#more part E
class TwoDArrayIterator:

    #### DO NOT CHANGE ##############
    def __init__(self, TDA):
        self.array = TDA.array
        self.i, self.j = 0, 0 #tracker for row index and column index
        self.m, self.n = TDA.shape()
    ################################

    def __next__(self):
        """
        Go through the array one number at a time.
        It should first go across the top row, then across the next row, then␣
 ↪across the next row, etc.
        Args:
            None
        Returns:
            the numbers in the array
        """

        if self.j ==  self.n:
            self.j = 0
            self.i += 1
            if self.i == self.m :
                raise StopIteration
        num = self.array[self.i][self.j]
        self.j += 1
        return num
```

7 Test for part A

[8]:
```
#This should run and not raise an error
x=TwoDArray([[1]])
y=TwoDArray([[1,1],[2,2]])
print(x)
print(y)
```

```
[[1]]
[[1, 1]
 [2, 2]]
```

[9]:
```
#this should raise an error
x=TwoDArray([(1,2)])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_14576/2866033377.py in <module>
      1 #this should raise an error
----> 2 x=TwoDArray([(1,2)])

~\AppData\Local\Temp/ipykernel_14576/522377246.py in __init__(self, array)
     13         ##########################
     14         if not isinstance(array, list) & all(isinstance(i, list) for i␣
 ↪in array):
---> 15             raise TypeError("This function is designed to work only wit␣
 ↪list of lists.")
     16         elif len(array)>1 and all(len(array[0]) != len(i) for i in␣
 ↪array[1:]):
     17             raise ValueError("All lists need to have the same length.")

TypeError: This function is designed to work only with list of lists.
```

[ ]:
```
#this should also raise an error
x=TwoDArray([1,2,3,4])
```

8 Test for Part B

[10]:
```
#this code should run error free
x=TwoDArray([[1,2],[3,4]])
print(2*x)
```

```
[[2, 4], [6, 8]]
```

9 Tests for Part C

[11]:
```
#this code should run error free
#(This should run even if you haven't implemented broadcasting)
x=TwoDArray([[0,0],[0,0]])
y=TwoDArray([[0,0],[0,0]])
z=TwoDArray([[1],[2]])
w=TwoDArray([[1,2]])
print(x+y)
```

```
[[0, 0]
 [0, 0]]
```

[12]:
```
#this code should run error free once you've implemented broadcasting
print(x+z)
print(z+x)
print(x+w)
print(w+x)
```

```
[[1, 1]
 [2, 2]]
[[1, 1]
 [2, 2]]
[[1, 2]
 [1, 2]]
[[1, 2]
 [1, 2]]
```

[13]:
```
# this should raise an appropriate error
print(x + [[1,1],[2,2]])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_14576/1409732978.py in <module>
      1 # this should raise an appropriate error
----> 2 print(x + [[1,1],[2,2]])

~\AppData\Local\Temp/ipykernel_14576/522377246.py in __add__(self, other)
     56         if not (isinstance(self, TwoDArray) & isinstance(other,␣
 ↪TwoDArray)):
     57             # if not, raise an appropriate error with a short error␣
 ↪message.
---> 58             raise TypeError ("Two instances should be TwoDArrays!")
     59         ############################
     60 

TypeError: Two instances should be TwoDArrays!
```

[14]:
```
#in this cell, show an example of where the arrays can't be broadcasting
#your solution should display an appropriate error messsage
a =TwoDArray([[0,1,2]])
b =TwoDArray([[0],[1],[2],[3]])
print(a + b)
```

```
This type of addition can work in numpy by broadcasting.
None
```

10 Tests for Part D

[15]:
```
#this code should run error free
x=TwoDArray([[0,0],[0,0]])
y=TwoDArray([[0,0],[0,0]])
z=TwoDArray([[1],[2]])
w=TwoDArray([[1,2]])
print(x-y)
```

```
[[0, 0]
 [0, 0]]
```

[16]:
```
#this code should run error free
print(x-z)
print(x-w)
print(z-x)
print(w-x)
```

```
[[-1, -1]
 [-2, -2]]
[[-1, -2]
 [-1, -2]]
[[1, 1]
 [2, 2]]
[[1, 2]
 [1, 2]]
```

11 Test for Part E

[17]:
```
x=TwoDArray([[1,2],[3,4]])
for t in x:
    print(t)
```

```
1
2
3
4
```

12 Problem 2 - 10 points

In problem 1, I told you to use `isinstance` rather than code of the form `type(x)==list`. In the box below, please explain the difference between these two things and one reason why it is often better to use `isinstance`. Your explanation should be up to five sentences long and be written in complete, grammatically correct sentences. Reminder: this is an open internet exam so googling "why should I use isinstance" is a perfectly valid strategy if you are not sure.

One reason is that isinstance is faster than the type method. Also, isinstance considers inheritance, which allows the user to check the certain type of a given object besides list, str, tuple, and so on, in other words, more flexible than the type method.

13 Problem 3 - 10 points

As you have noticed, Python is quite different from C++. In boxes below, please briefly describe a) at least two advantages of Python and b) at least two disadvantages. Your solution for each part should be 2 to 6 senteces. Please write in complete, grammatically correct sentences. (Again, if you are not sure what to write, googling is a valid strategy here.)

Advantages:

```
1. Python is better for beginners in terms of its easy-to-read code and simple syntax.
2. Python is a leading language for data analysis and machine learning, wide range of applicati
3. Huge standard library. Just to pick some random examples, Python ships with several XML pars
4. Rapid Prototyping is possible because of the small size of the code
```

Disadvantages:

```
1. C++ performs efficiently and the speed is faster when compared to Python.
2. C++ is suitable for almost every platform including embedded systems whereas Python can be u
3. C++ is more predictable than Python which is dynamically typed.
4. C++ is the language is closer to hardware over Python.
```

13.0.1 Example of a function calling itself

The function below simulates absolute value

[18]:
```
def f(x):
    if x>=0:
        return x
    else:
        return(f(-x))
```

[ ]:
```
print(f(8), f(-8))
```

[ ]:
```
```