Homework 1

35 points
Due Thursday, February 13, 2025 at 11:59 PM

Vectorized Matrix Addition Benchmarking (10 Points)

Matrix addition is the process of adding corresponding elements of two matrices together. As an example, consider addition with two 2×3 matrices a and b:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix} + \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix} = \begin{bmatrix} a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} & a_{1,3} + b_{1,3} \\ a_{2,1} + b_{2,1} & a_{2,2} + b_{2,2} & a_{2,3} + b_{2,3} \end{bmatrix}$$

The result is a 2×3 matrix where the value at i, j equals $a_{i,j} + b_{i,j}$.

Using Python data structures, we can represent any 2-dimensional $m \times n$ matrix as a list of m n-element lists. For example, here are two 2×3 matrices:

```
matrix_a = [
    [4, 2, 6],
    [1, 9, 0]
]
```

```
matrix_b = [
    [1, 0, 0],
    [4, 0, 3]
]
```

Their sum is the following:

```
[
    [5, 2, 6],
    [5, 9, 3]
]
```

Write a Python function `matrix_add_2d(a, b)` that returns the sum of two 2-dimensional lists like `matrix_a` and `matrix_b`. Do not use NumPy or NumPy arrays in your implementation. Only use native Python data structures and loops. You may assume the two lists are of equal size.

Benchmark your function using the `%timeit` command like we did in the "Lecture 1: Why NumPy?" notebook available in Blackboard. Try benchmarking it with small, medium, and very large matrices. Compare the performance of your function to NumPy's vectorized addition functionality, which you can access by using the `+` operator with your two NumPy arrays, or by calling the `np.add()` ufunc with the two arrays as arguments. Report what you find in a written submission.

Documentation for `np.add()`: https://numpy.org/doc/stable/reference/generated/numpy.add.html

Vectorized Matrix Addition Written Questions (5 points)

1. (2 points) Besides the speed difference, can you think of any other reasons to prefer NumPy's adddition ufunc over your handwritten matrix addition function?

2. (3 points) `np.add()` checks the dimensions of arrays to ensure it is possible to add them together. If you wanted to check that your 2-dimensional lists are the same size, how would you have to modify your program and what would you have to check? As part of your answer, give an error condition that could occur with 2-dimensional lists but not with NumPy arrays.

Broadcasting (10 points)

Use the 3 array broadcasting rules to determine whether each of the following pairs of arrays can be added together. For each pair, show how each of the rules are applied and how they affect the shape of each array. Verify your answer by performing the addition in NumPy.

Example

```
a = np.array([
    [4, 2, 6]
])

b = np.array([4])
```

1. Start: `a.shape` = (1, 3), `b.shape` = (1,).
2. Apply Rule 1 to `b`: `a.shape` = (1, 3), `b.shape` = (1, 1).
3. Apply Rule 2 to `b`: `a.shape` = (1, 3), `b.shape` = (1, 3).
4. Perform elementwise addition.

Result of `a + b`:

```
np.array([
    [8, 6, 10]
])
```

Problems

1.
```
a = np.array([
    [1, 1, 2],
    [3, 2, 1],
    [1, 9, 1]
])

b = np.array([
    [1],
    [2],
    [3]
])
```

2.
```
a = np.array([
    [
        [1, 2],
        [3, 1]
    ],
    [
        [8, 7],
        [9, 1]
    ]
])

b = np.array([10, 11])
```

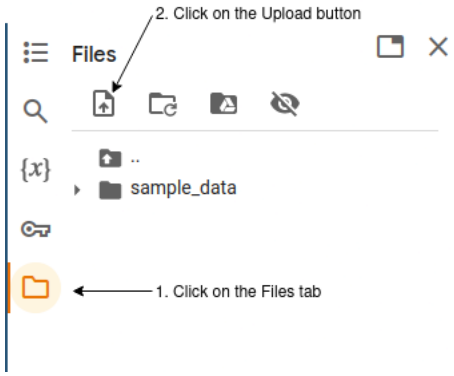
3.
```
a = np.array([1, 2, 3])

b = np.array([
    [10],
    [11],
    [12]
])
```

4.
```
a = np.array([
    [
        [1, 2],
        [3, 1]
    ],
    [
        [8, 7],
        [9, 1]
    ]
])

b = np.array([10, 11, 12, 13])
```

5.
```
a = np.array([
    [
        [
            [1],
        ]
    ],
    [
        [
            [2]
        ]
    ]
])

b = np.array([3, 4])
```

Answering Questions with NumPy (10 points)

A file uploaded with this assignment, `solar.txt`, contains simulated data from the power output of a residential solar panel installation. To use this data, you can upload it to Colab using its file storage:



Once the file has successfully uploaded, you can load it to a NumPy array with the following code:

```
solar_data = np.loadtxt('solar.txt')
```

Once you load the solar data, examine its size. The first dimension represents days of the year. The second dimension represents the number of watts generated in the morning, afternoon, and evening, respectively.

Using NumPy functionality we discussed in class, including slicing, relational operators, masking, and NumPy ufuncs, answer the following questions. Do not use Python functionality like loops or lists. Each question can be answered with a single statement (note that some problems may include more than one question). When necessary, use the `axis` parameter for ufuncs like `mean()`.

In cases where the answer involves multiple numbers, please label them in your writeup so it is clear what it represents.

1. How many watts were generated all year?
2. What were the most watts generated in the morning, afternoon, and evening? The least?
3. What is the average number of watts generated on January afternoons (i.e, afternoons for the first 31 days)?
4. How many days did the system generate fewer than 500 watts total (in the morning, afternoon, and evening)?
5. What is the average number of watts generated in the morning, afternoon, and evening all year? Based on what you find, which direction do you believe the solar panels are facing?

Submission Instructions

In Blackboard, submit written responses in an appropriate text format (PDF, Word, LibreOffice, etc) and your *.ipynb file(s). Do not submit a share link.