hw02_teng_gradescope

September 26, 2019

0.1 Homework 2

Please import the following package.

[1]:
```
import numpy as np
```

0.1.1 Loops

How to work with multiple indices?

Use the code from Homework 1 Question 1 to write a function `flatten` that takes as input an array with shape `(r,c)` and outputs an array with shape `(rc,)`. For example, input `np.array([[1,2], [3,4]])` would yield output `np.array([1,2,3,4])`. Note that the array has two indices. The indices indicate the row and column. We flatten the array by going across the columns in each row.

[2]:
```
# Implement this
outputArray = []

def flatten(arr):
    for row in arr:
        for elem in row:
            outputArray.append(elem)
    return(outputArray)

#     raise NotImplementedError()
```

[3]:
```
#arr =[[1,2], [3,4]]
arr = np.array([[1,2], [3,4]])
flatten(arr)
```

[3]:
```
[1, 2, 3, 4]
```

[4]:
```
arr.dtype
```

[4]:
```
dtype('int64')
```

Modify the function from Question 1.
Add a parameter called `major`.
Make the default value of `major` be `"row"`
Rewrite the loop so
If `major` is `"row"`, then the array is flattened in row major order

If `major` is `"column"`, then the array is flattened in column major order

[5]:
```
# Implement this, including modifying arguments
arr = np.array([[1,2,3],[4,5,6]])

def flatten_v2(arr, major="row"):
    outputArray = []
    row, col = arr.shape
    if major == "row":
        for i in range(row):
            for j in range(col):
                outputArray.append(arr[i][j])
    if major == "column":
        for i in range(col):
            for j in range(row):
                outputArray.append(arr[j][i])
    return(outputArray)

print(flatten_v2(arr))
print(flatten_v2(arr, "column"))
```

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

Modify the function from Question 1 to allow for more than two indices. For example, if the input is an array with shape `(r,c,h)` then the output is an array with shape `(rch,)`. So input `np.array([ [[1,2], [3,4]], [[5,6], [7,8]] ])` which has shape `(2,2,2)` would yield output `np.array([1,2,3,4,5,6,7,8])` which has shape `(8,)`. You should approach the problem using recursion.

[6]:
```
# Implement this

#iterative solution
def flatten_v3(arr):
    #raise NotImplementedError()
    outputArray = []
    row, col, height = arr.shape
    for i in range(height):
        for j in range(row):
            for k in range(col):
                outputArray.append(arr[i][j][k])
    return(outputArray)

arr = np.array([ [[1,2], [3,4]], [[5,6], [7,8]] ])
flatten_v3(arr)
```

[6]:
```
[1, 2, 3, 4, 5, 6, 7, 8]
```

[33]:
```
#recursive solution
def flatten_v3(arr):
```

```
    #raise NotImplementedError()
    outputArray = []
#     height, row, col = arr.shape

    if len(arr.shape)==1:
        return(arr)

    while len(arr.shape) > 1:
        for i in arr:
            outputArray.extend(flatten_v3(i))
            #or if you just want it as a outputArray you can remove np.array
        return(np.array(outputArray))

arr = np.array([ [[1,2], [3,4]], [[5,6], [7,8]] ])

flatten_v3(arr)
```

[33]:
```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

[34]:
```
flatten_v3(arr).shape
```

[34]:
```
(8,)
```

0.1.2 Storage

How to compress an array with lots of zeros

Write a function called `dense_to_sparse` that inputs an array and outputs a dictionary with
Keys as tuples containing `(row,column)` of all non-zero entries
Values as the corresponding non-zero entries.
The resulting entries should be in row-major order.
For example, if the input is `np.array([[1,0], [0,4]])` then the output is `{(0,0):1, (1,1):4}`

[14]:
```
# Implement this
def dense_to_sparse(arr):
    #raise NotImplementedError()
    outputDictionary = {}
    row, col = arr.shape
    for i in range(row):
        for j in range(col):
            if arr[i][j] != 0:
                outputDictionary.update({(i,j) : arr[i][j]})
    return(outputDictionary)

arr = np.array([[1,0], [0,4]])
# arr = np.array([[1,0,3], [0,4,3], [0,3,3]])
print(dense_to_sparse(arr))
```

```
{(0, 0): 1, (1, 1): 4}
```

Write an inverse function called `sparse_to_dense`

[28]:
```
# Implement this
def sparse_to_dense(arr):
    #initialize matrix with all 0's w size of dict
    max_row_num=0
    max_col_num = 0
    for (i,j) in arr:
        if i > max_row_num:
            max_row_num = i
        if j > max_col_num:
            max_col_num = j
    total_rows = max_row_num + 1
    total_cols = max_col_num + 1

    new_dict=np.zeros([total_rows, total_cols])

    #indexing
    for (i,j) in arr:
        new_dict[i][j] = arr[(i,j)]
        #if you want output as a list use tolist, otherwise keep it as an array
        #new_dict.tolist()
    return(new_dict)

arr = {(0, 0): 1, (1, 1): 4}
sparse_to_dense(arr)
```

[28]:
```
array([[1., 0.],
       [0., 4.]])
```

[ ]:
```
```