Chap 4 - Transpose in NumPy

March 24, 2018

0.1 Transpose in NumPy

In NumPy, we can obtain the transpose of a matrix by accessing its `T` attribute or using a `transpose()` function.

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

        m = np.array([[1,2],[10,20],[100,200]]) # 3x2
        print('m:\n', m)
        print('transpose of m:\n', m.T) # m.T is the transpose of m
```

```
m:
 [[  1   2]
 [ 10  20]
 [100 200]]
transpose of m:
 [[  1  10 100]
 [  2  20 200]]
```

Note that no data in the memory is moving. It simply changes the way it indexes the original matrix. That is `m` and `m.T` are **sharing the same data**.

```
In [2]: m_t = m.T
        m_t[1][0] = -2 # m_t[1][0] = 2
        print('m_t:\n', m_t)
        print('m:\n', m)
```

```
m_t:
 [[  1  10 100]
 [ -2  20 200]]
m:
 [[  1  -2]
 [ 10  20]
 [100 200]]
```

0.1.1 A real use case in neural network

If we have two matrices called `inputs` and `weights`

```
In [3]: inputs = np.array([[1,2,3,4]])
        print('inputs:\n', inputs)
        print('inputs shape:', inputs.shape)

        weights = np.array([[0.1,0.2,0.3,0.4],\
                            [-0.1,-0.2,-0.3,-0.4],\
                            [0.01, 0.02, 0.03, 0.04]]) # 3x3
        print('weights:\n', weights)
        print('weights shape:\n', weights.shape)
```

```
inputs:
 [[1 2 3 4]]
inputs shape: (1, 4)
weights:
 [[ 0.1   0.2   0.3   0.4 ]
 [-0.1  -0.2  -0.3  -0.4 ]
 [ 0.01  0.02  0.03  0.04]]
weights shape:
 (3, 4)
```

What can we do if we want to find the matrix product of these two matrices? This is because with their current shape, they are incompatible.

```
In [4]: np.matmul(inputs, weights)
```

```
        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-4-6e050fb6601d> in <module>()
    ----> 1 np.matmul(inputs, weights)


        ValueError: shapes (1,4) and (3,4) not aligned: 4 (dim 1) != 3 (dim 0)
```

What if we transpose the weight matrix. (Note: this works!)

```
In [5]: inputs = np.array([[1,2,3,4]])
        print('inputs:\n', inputs)
        print('inputs shape:', inputs.shape)

        weights = np.array([[0.1,0.2,0.3,0.4],\
```

```
                            [-0.1,-0.2,-0.3,-0.4],\
                            [0.01, 0.02, 0.03, 0.04]]) # 3x4
        weights_t = weights.T
        print('weights:\n', weights)
        print('weights_t:\n', weights_t)
        print('weights_t shape:\n', weights_t.shape)

        # Then compute the matrix product
        results = np.matmul(inputs, weights.T)
        print('results:\n', results)
```

```
inputs:
 [[1 2 3 4]]
inputs shape: (1, 4)
weights:
 [[ 0.1   0.2   0.3   0.4 ]
 [-0.1  -0.2  -0.3  -0.4 ]
 [ 0.01  0.02  0.03  0.04]]
weights_t:
 [[ 0.1  -0.1   0.01]
 [ 0.2  -0.2   0.02]
 [ 0.3  -0.3   0.03]
 [ 0.4  -0.4   0.04]]
weights_t shape:
 (4, 3)
results:
 [[ 3.  -3.   0.3]]
```

Another possible solutions is by taking the *transpose* of `inputs` (i.e. `4x1`) then swap their order. (Note, this also works)

```
In [6]: weights = np.array([[0.1,0.2,0.3,0.4],\
                            [-0.1,-0.2,-0.3,-0.4],\
                            [0.01, 0.02, 0.03, 0.04]]) # 3x4
        print('weights:\n', weights)
        print('weights shape:\n', weights.shape)

        inputs = np.array([[1,2,3,4]])
        inputs_t = inputs.T
        print('inputs:\n', inputs)
        print('inputs_t:\n', inputs_t)
        print('inputs_t shape:', inputs_t.shape) # 4x1

        # Then compute the matrix product
        results = np.matmul(weights, inputs.T)
        print('results:\n', results)
```

```
weights:
 [[ 0.1   0.2   0.3   0.4 ]
```

```
 [-0.1  -0.2  -0.3  -0.4 ]
 [ 0.01  0.02  0.03  0.04]]
weights shape:
 (3, 4)
inputs:
 [[1 2 3 4]]
inputs_t:
 [[1]
 [2]
 [3]
 [4]]
inputs_t shape: (4, 1)
results:
 [[ 3. ]
 [-3. ]
 [ 0.3]]
```

Both solutions work, so which solution to choose is depends on how we want the shape of the output to be.