Notes on Tensor Axes Reordering

In the boot camp today, we saw a 3-dimensional tensor of shape (4, 3, 2) whose axes were arranged in the order as follows:

(1) 0th axis of size 4 corresponded to time stamps;
(2) 1st axis of size 3 corresponded to patients;
(3) 2nd axis of size 2 corresponded to features.

In summary, the tensor stored information of 2 features for 3 patients across 4 time stamps. Note how the description here starts from the innermost axis (features) to the outermost axis (time stamps). The actual tensor example we saw was

```
T = np.array([[[74, 128], [79, 116], [71, 116]],
              [[78, 118], [82, 124], [72, 128]],
              [[84, 138], [84, 130], [74, 120]],
              [[82, 126], [76, 156], [82, 132]]])
print(T.shape)
print(T)
```

```
(4, 3, 2)
[[[ 74 128]
  [ 79 116]
  [ 71 116]]

 [[ 78 118]
  [ 82 124]
  [ 72 128]]

 [[ 84 138]
  [ 84 130]
  [ 74 120]]

 [[ 82 126]
  [ 76 156]
  [ 82 132]]]
```

The next task we wanted to do was to reshape the tensor such that we had the axes arranged in the following order:

(1) 0th axis of size 3 corresponded to patients;
(2) 1st axis of size 2 corresponded to features;
(3) 2nd axis of size 4 corresponded to time stamps.

That is, the resulting tensor should store information across 4 time stamps for 2 features for 3 patients. Again, note how the description here starts from the innermost axis (timestamps) to the outermost axis (patients). The resulting tensor will have shape (3, 2, 4).

In order to achieve this, many of you today used the np.reshape() function as follows:

```
print(T.reshape(3, 2, 4))
```

```
[[[ 74 128  79 116]
  [ 71 116  78 118]]

 [[ 82 124  72 128]
  [ 84 138  84 130]]

 [[ 74 120  82 126]
  [ 76 156  82 132]]]
```

Note, however, that the result is not what we wanted even though the resulting tensor has shape (3, 2, 4). To confirm this, consider printing T[0] after reshaping, which should correspond to the 0th element w.r.t. the outermost axis (patient-axis), or should give us the 0th patient's information across all features and time stamps. Doing so results in the following:

```
T_reshaped = T.reshape(3, 2, 4)
print(T_reshaped[0])
```

```
[[ 74 128  79 116]
 [ 71 116  78 118]]
```

This clearly is not the 0th patient's information across all features and time stamps. How do when achieve the desired result? For that, we have to use the np.transpose() function as follows:

```
T_reshaped = T.transpose(1, 2, 0)
print(T_reshaped)
```

```
[[[ 74  78  84  82]
  [128 118 138 126]]

 [[ 79  82  84  76]
  [116 124 130 156]]

 [[ 71  72  74  82]
  [116 128 120 132]]]
```

Note that, printing T[0] after transposing results in the 0th patient's information across all features and time stamps as follows:

```
T_reshaped = T.transpose(1, 2, 0)
print(T_reshaped[0])
```

```
[[ 74  78  84  82]
 [128 118 138 126]]
```

How do we understand this? Simply note that we specify the shuffling of the axes as the argument to the np.transpose() function. That is, we go from (time stamps, patients, features) to (patients, features, timestamps), or we go from the axes order (0, 1, 2) to (1, 2, 0).

Now, try this as an exercise: reorder the tensor T such that the axes are arranged in the following order:

(1) 0th axis of size 3 corresponded to patients;
(2) 1st axis of size 4 corresponded to time stamps;
(3) 2nd axis of size 2 corresponded to features.

What does the output of T_reshaped[0] indicate now? What does the output of T_reshaped[:, :, 1] indicate?